

Rel. date 2-3-93
SBIK-14.02.2250

P.305

MULTI-ACCESS LASER COMMUNICATIONS TERMINAL

FINAL REPORT

CONTRACT NO. NAS5-31170

Submitted to:

NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

Laser Data Technology, Inc.
Saint Louis, Missouri
November 3, 1992

(NASA-CR-191251) MULTI-ACCESS
LASER COMMUNICATIONS TERMINAL Final
Report (Laser Data Technology)
305 p

N94-23830

Unclass

G3/32 0204535

TABLE OF CONTENTS

	PAGE
1.0 EXECUTIVE SUMMARY	1
2.0 DESIGN OVERVIEW	6
2.1 Optical	6
2.1.1 Telescope	6
2.1.2 Focal-plane Optics	6
2.2 Mechanical	6
2.2.1 Disk and Arm Assemblies	6
2.3 Acquisition and Tracking Design Constraints	14
2.4 Electronics	22
2.4.1 Motion Control Circuitry	22
2.4.2 Communications Circuits	31
2.5 Software	31
2.5.1 Executive	35
2.5.2 Angle Processor Software	40
2.5.3 Motion Processor Software	40
3.0 EVALUATION TEST DATA	42
3.1 Optical Tests	42
3.1.1 Telescope Performance	42
3.1.2 Arm Performance	43
3.2 Mechanical Tests	43
3.2.1 Resolution/backlash/hysteresis	43
3.2.2 Slewing Rate	49
3.3 Acquisition Time	54
3.4 Tracking Error	54
3.5 Weight Analysis	54
4.0 AREAS FOR FURTHER DEVELOPMENT	58
5.0 CONCLUSIONS	59
APPENDIX A TELESCOPE OPTICAL DESIGN SPECIFICATIONS	A1-A??
APPENDIX B COMPLETE ELECTRONIC SCHEMATICS	B1-B??
APPENDIX C COMPLETE SOFTWARE LISTINGS	C1-C??

LIST OF FIGURES

	PAGE
1-1 Multi-access Laser Communication System	2
1-2 Prototype Terminal	2
1-3 Movable Image Pickup System (MIPS) Concept	3
1-4 Disk and Arm Rotation System	4
1-5 Goals of Multi-access Terminal	4
2.1-1 Glass Telescope Elements for Space Design	6
2.1-2 Glass Telescope Spot Diagrams - Positions 1 & 2	7
2.1-3 Glass Telescope Spot Diagrams - Positions 3 & 4	8
2.1-4 Plastic Telescope Elements for Prototype Design	9
2.1-5 Use of Plastic Elements for Prototype System	10
2.1-6 Movable Image Pickup Optics	11
2.1-7 Completed Arm Assembly	12
2.2-1 Arm and Disk Coordinates in the Telescope Field of View	14
2.2-2 Worm Drive Assembly	15
2.2-3 Disk Support Structure	16
2.2-4 Disk Assembly End Plate	17
2.2-5 Disk/Arm Resolution	18
2.3-1 Acquisition Approach	20
2.4-1 System Electronics Layout	22
2.4-2 Motor Controller Chip Block Diagram	23
2.4-3 Total Control Loop	24
2.4-4 Motor and Total Loop Responses	25
2.4-5 Motor Switching Schematic for Channel 2	27
2.4-6 Quadrant Detector Electronics Schematic	28
2.4-7 Receiver Waveform Showing Data and Tone	29
2.4-8 Angle Processor Electronics Schematic	31
2.4-9 Wideband Amplifier Schematic	32
2.4-10 Laser Driver Circuit Schematic	33
2.5-1 Control Software Top-level Architecture	36
2.5-2 Error Signal Amplitude Detection	40
3.1-1 Elevation Quadrant Scan	43
3.1-1 Azimuth Quadrant Scan	44
3.1-3 Spot Shape Measurement	45
3.2-1 Hysteresis Plot - Before Adjustment	46
3.2-2 Hysteresis Plot - After Adjustment	47
3.2-3 Backlash Test	49
3.2-4 Spot Shape Measurement	50
3.2-5 Motor Step Response - Before Cleaning	51
3.2-6 Motor Step Response - After Cleaning	52
3.4-1 Tracking Error Parallel to Arm	54
3.4-2 Tracking Error Perpendicular to Arm	55
3.5-1 Multiple-Access Terminal Weight Analysis	56

LIST OF TABLES

	PAGE
2.3-1 Acquisition Error Budget	20
2.5-1 Manual Operation Software Organization	35
2.5-2 Manual Control Menu	35
2.5-3 Operational Satellite Software Organization	37
2.5-4 Automatic Real-time Operation Software Organization	38
2.5-5 Angle Processing Routines	39

1.0 EXECUTIVE SUMMARY

The Optical Multi-Access (OMA) Terminal is capable of establishing up to six simultaneous high-data-rate communication links between low-Earth-orbit satellites and a host satellite at synchronous orbit with only one 16-inch-diameter antenna on the synchronous satellite. The advantage over equivalent RF systems in space weight, power, and swept volume is great when applied to NASA satellite communications networks (Figure 1-1).

Figure 1-2 is a photo of the 3-channel prototype constructed under the present contract to demonstrate the feasibility of the concept. The telescope has a 10-inch clear aperture and a 22° full field of view. It consists of 4 refractive elements to achieve a telecentric focus, i.e., the focused beam is normal to the focal plane at all field angles. This feature permits image pick-up optics in the focal plane to track satellite images without tilting their optic axes to accommodate field angle. Figure 1-3 shows the geometry of the image-pick-up concept, and Figure 1-4 shows the coordinate system of the swinging arm and disk mechanism for image pick-up. Optics in the arm relay the telescope focus to a communications and tracking receiver and introduce the transmitted beacon beam on a path collinear with the receive path. The electronic circuits for the communications and tracking receivers are contained on the arm and disk assemblies and relay signals to an associated PC-based operator's console for control of the arm and disk motor drive through a flexible cable which permits $\pm 240^\circ$ travel for each arm and disk assembly. Power supplies and laser transmitters are mounted in the cradle for the telescope. A single-mode fiber in the cable is used to carry the laser transmitter signal to the arm optics. Figure 1-5 shows the promise of the optical multi-access terminal towards which the prototype effort worked. The emphasis in the prototype development was the demonstration of the unique aspects of the concept, and where possible, cost avoidance compromises were implemented in areas already proven on other programs.

The design details are described in Section 2, the prototype test results in Section 3, additional development required in Section 4, and conclusions in Section 5.

MULTI-ACCESS LASER COMMUNICATIONS SYSTEM

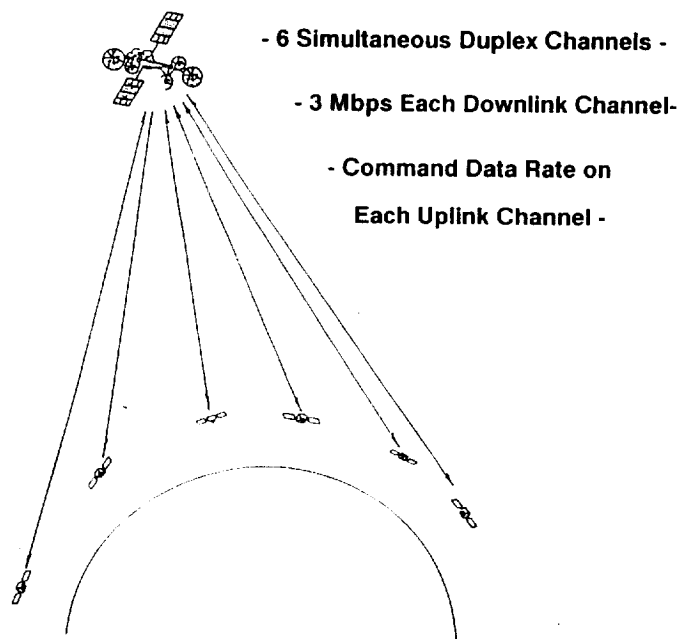


FIGURE 1-1

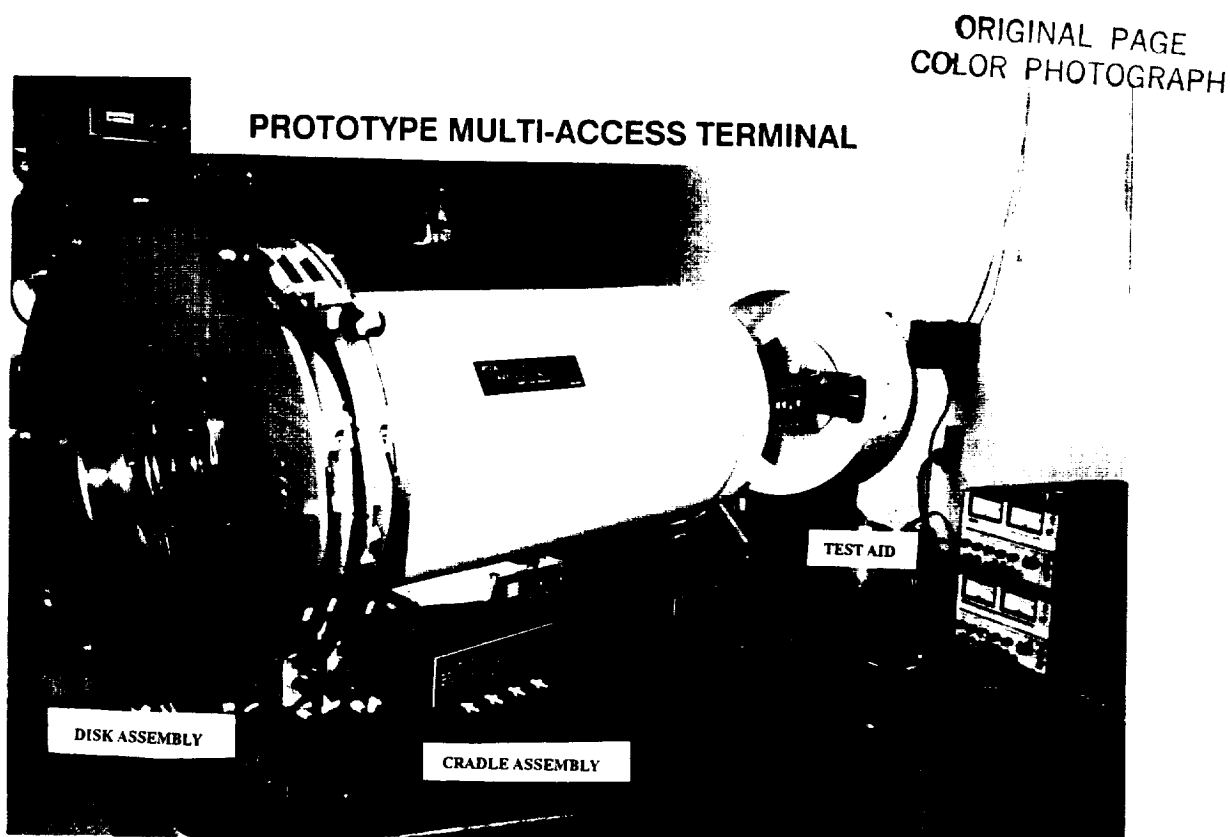
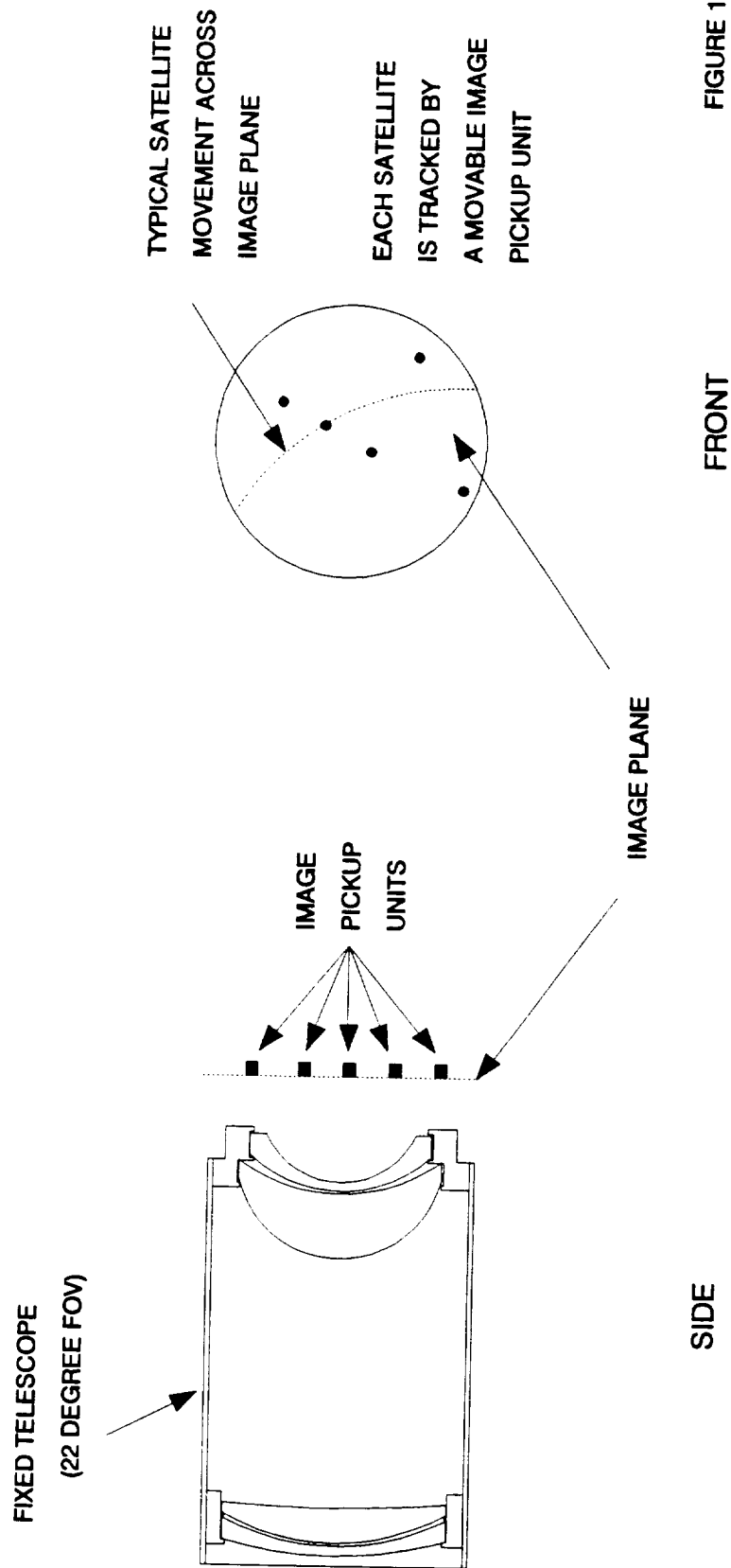


FIGURE 1-2

MOVABLE IMAGE PICKUP SYSTEM (MIPS) CONCEPT

EACH ANGULAR POINT IN THE FIELD OF VIEW IS REPRESENTED BY A POINT IN THE IMAGE PLANE



DISK AND ARM ROTATION SYSTEM

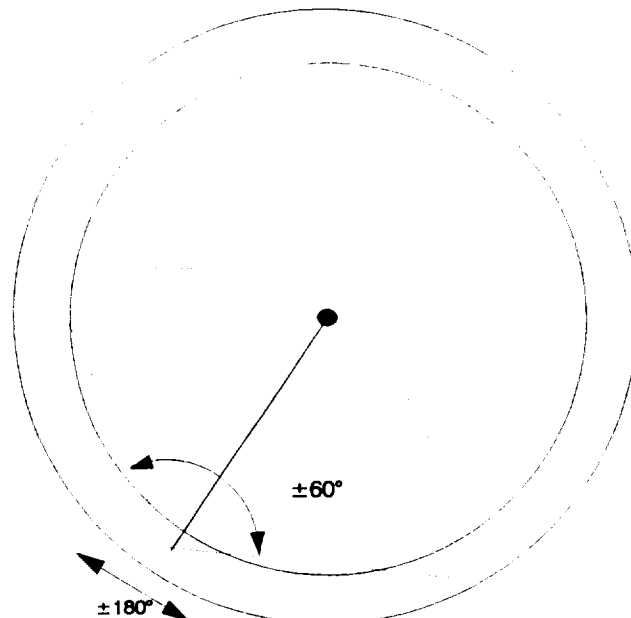


FIGURE 1-4

DISK ROTATES $\pm 180^\circ$, ARM ROTATES $\pm 60^\circ$
ARM ATTACHED TO DISK

SIX DISK ASSEMBLIES SPACED VERTICALLY SERVE SIX CHANNELS

GOALS OF BASELINE MULTI-ACCESS TERMINAL

- SIMULTANEOUS COMMUNICATION WITH SIX INDEPENDENT ASYNCHRONOUS LINKS
- FULL COVERAGE OF EARTH AND LOW ORBITS
- LIGHT WEIGHT (≈ 150 POUNDS)
- SMALL SIZE (10-INCH-DIAMETER CLEAR APERTURE)
- LOW POWER (< 100 WATTS)
- LOW SWEEP VOLUME (NO GIMBALS)
- 3 MBPS PER COMM LINK (CONCEPT SUPPORTS HIGHER RATE)
- 10 KBPS FORWARD RATE PER LINK

FIGURE 1-5

2.0 DESIGN OVERVIEW

2.1 Optical

2.1.1 Telescope - The telescope elements are shown in Figure 2.1-1 for the space design. Radiation hardened glass is specified for long life in space. Figure 2.1-2 and -3 show the spot diagram at the focal plane for various off-axis positions with a 300- μm circle for reference. This diameter represents 480 μrad in the far field of the system. Figure 2.1-4 depicts the plastic design used in the prototype to save cost. This material is unsuitable for space due to darkening effects of radiation but provides comparable optical performance for demonstration in the laboratory. Figure 2.1-5 compares the space and prototype designs. Appendix A contains the detailed lens specifications for both the glass and plastic designs.

2.1.2 Focal-plane optics - Figure 2.1-6 shows the arrangement of optics for the space design. The figure is arranged with the arms aligned one over the other in a vertical plane and show the locations of the channels with respect to the telescope focal plane. The receive channels are designed to relay the focus to the quadrant detector through a narrow-band optical filter for transmitter rejection. The transmitter signals are introduced by means of a single-mode fiber whose image is formed at the telescope image plane and coaligned with the arm receiver optical axis. The beamwidth is 150 μrad FWHM, so the image is 94 μm , FWHM. Because the telescope forms a telecentric image, the arms need only be located at the correct spot to receive the light, and no tilt of the optic axis as a function of field angle is required. In the prototype we included Channel 1, a channel between 2 and 3, and Channel 4, renamed Channels 1, 2, and 3, respectively. Figure 2.1-7 is a photograph of a complete arm for Channel 3 fully assembled with optics and electronics.

2.2 Mechanical

2.2.1 Disk and arm assemblies - The arm is mounted on a 354-tooth worm-gear ring and attached to a second worm-gear ring. Moving the two rings together moves the arm pivot point through more than one complete revolution, while differential movement of the rings with respect to each other swings the arm on its pivot. Therefore, by driving the worm on each ring (or "disk", as we have referred to them previously) to the proper location, any point in the field of the telescope can be

02-14-1991

LDT NASA TELESCOPE AT 820NM WAVELENGTH

13: 14: 41

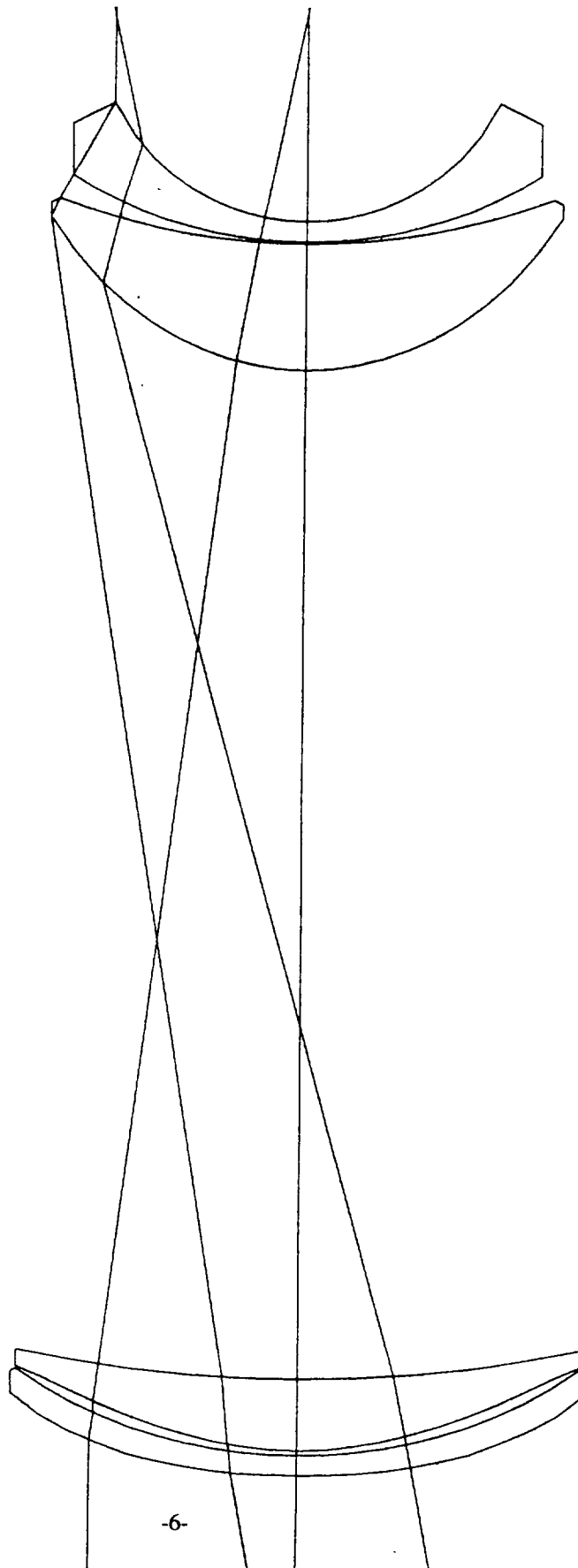
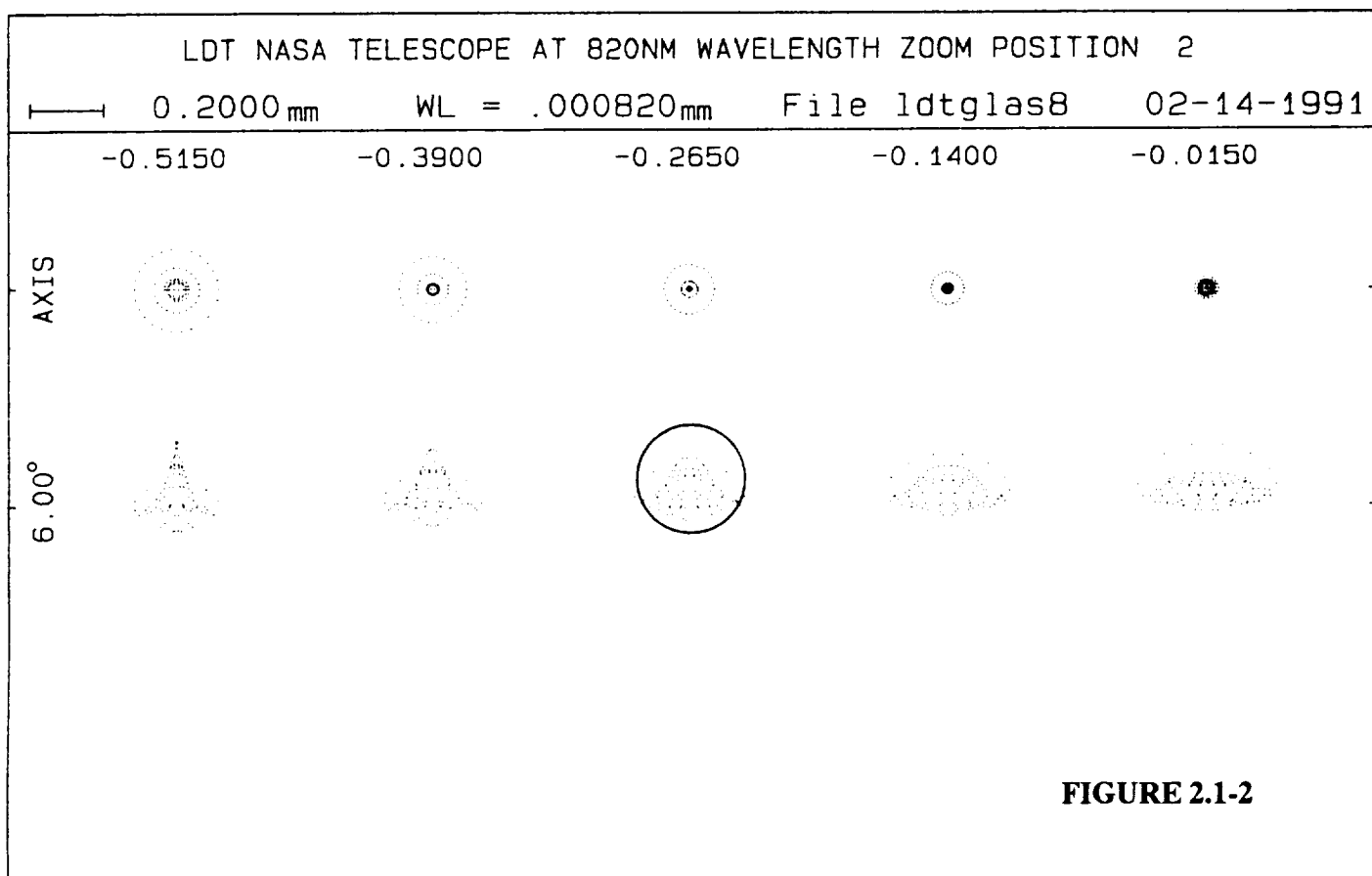
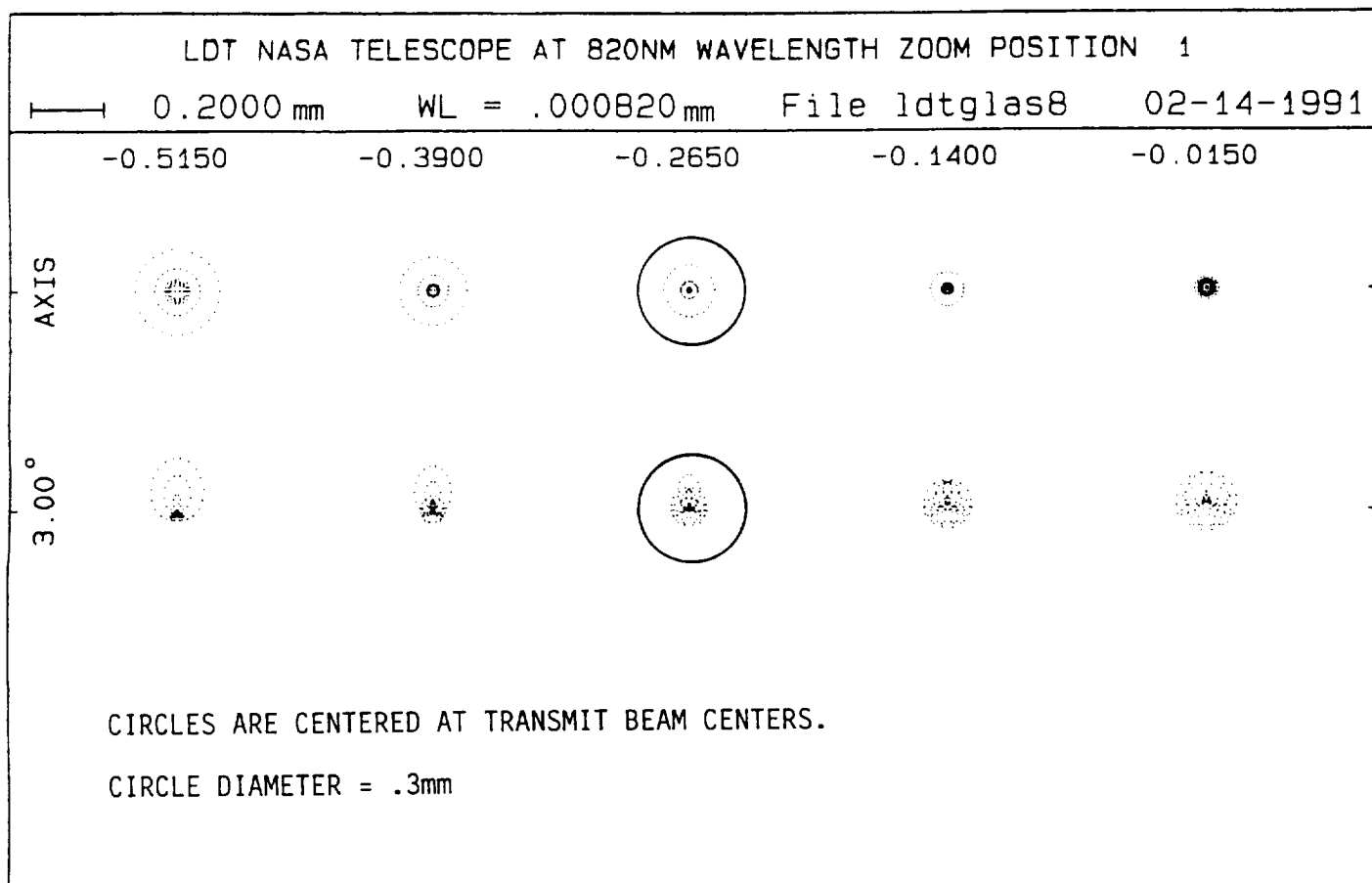


FIGURE 2.1-1

File ldtglas8

Zoom Position 4

SCALE = 0.25



MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

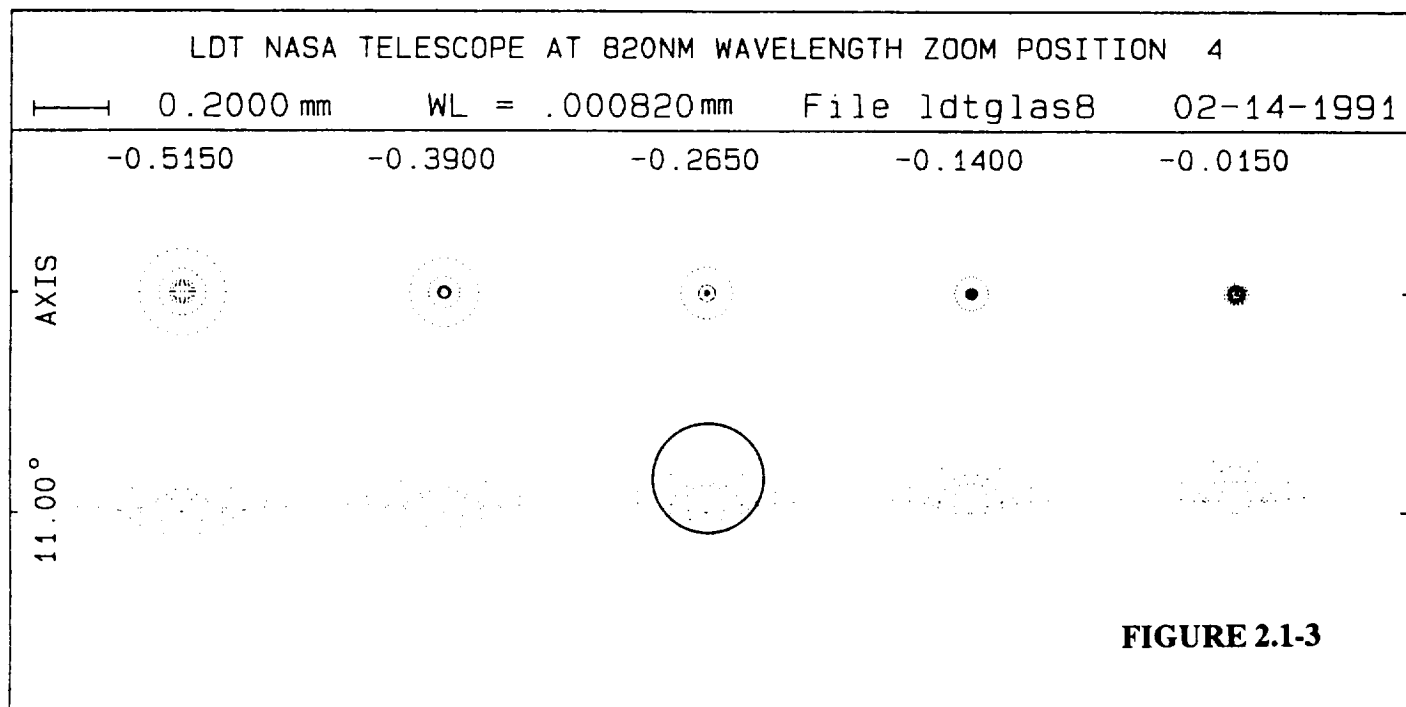
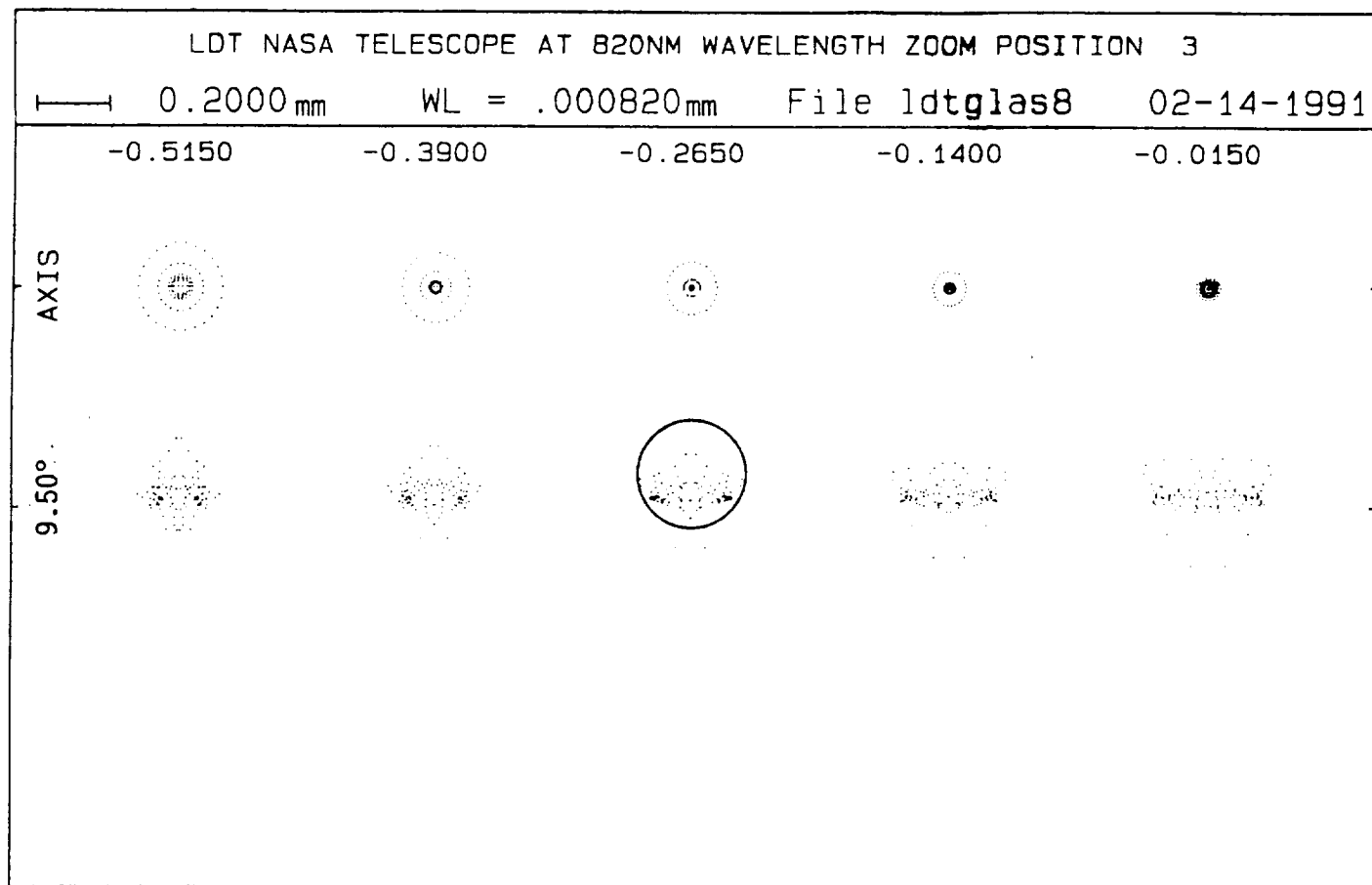


FIGURE 2.1-3

LDT NASA TELESCOPE (PLASTIC DESIGN)

09-04-1990

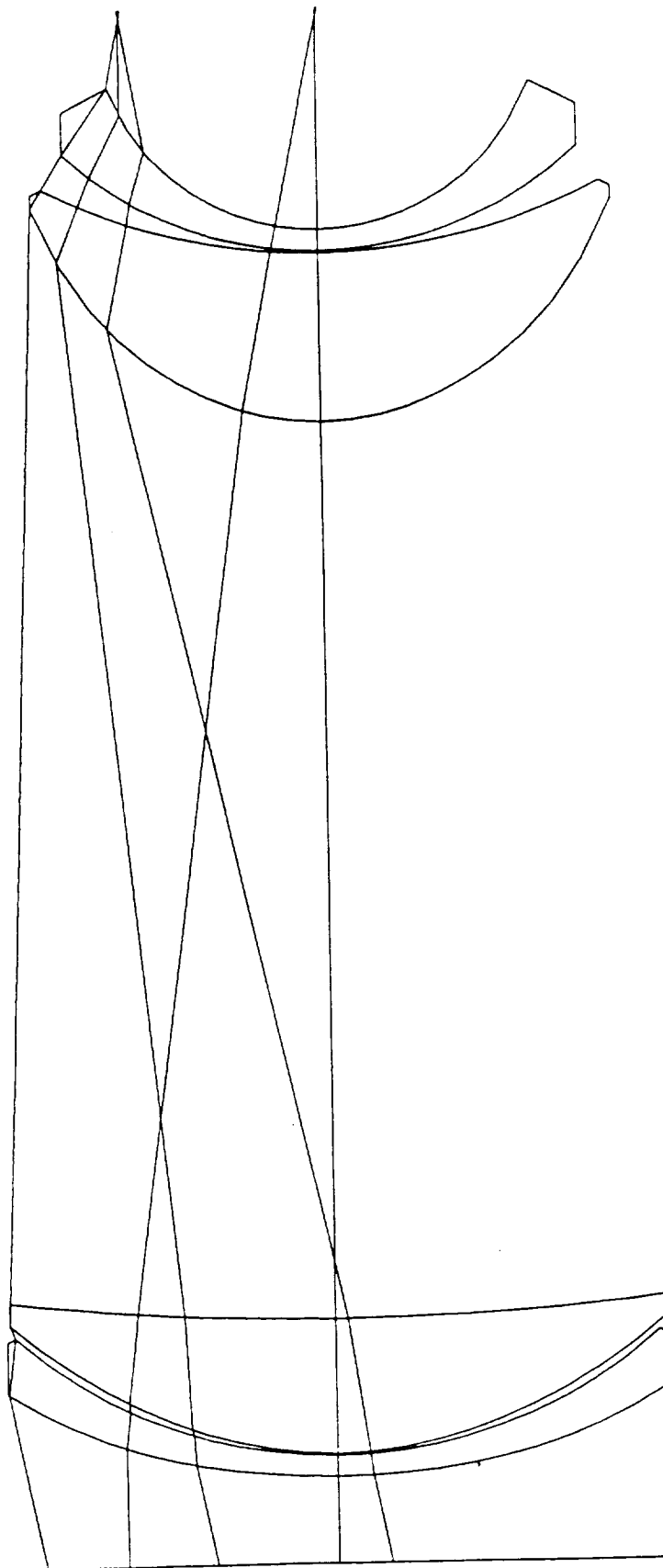


FIGURE 2.1-4

SCALE = 0.2

File ldtplas

USE OF PLASTIC ELEMENTS FOR PROTOTYPE SYSTEM

PARAMETER	SIMILARITY TO SPACE DESIGN
FUNCTIONAL PERFORMANCE	✓
SIZE	SLIGHTLY LARGER FOR EQUIVALENT PERFORMANCE
FIELD OF VIEW	✓
BACKSCATTER	✓
ALIGNMENT SENSITIVITY	✓
MINIMUM SPOT SIZE	CLOSE
POLARIZATION PRESERVATION	NOT SIMILAR
WEIGHT	LIGHTER
THERMAL MISALIGNMENT	NOT SIMILAR

FIGURE 2.1-5

MOVABLE IMAGE PICKUP OPTICS

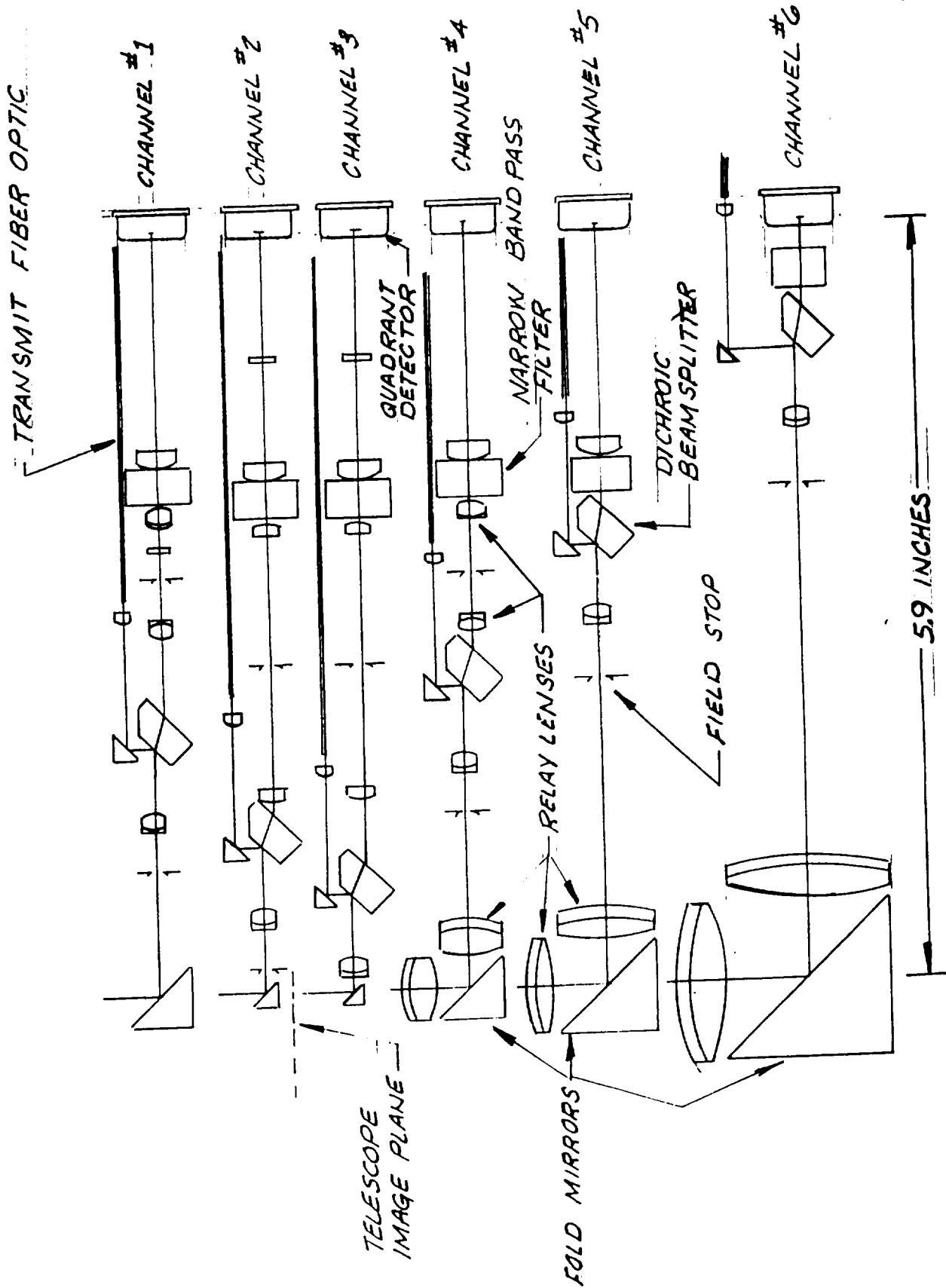
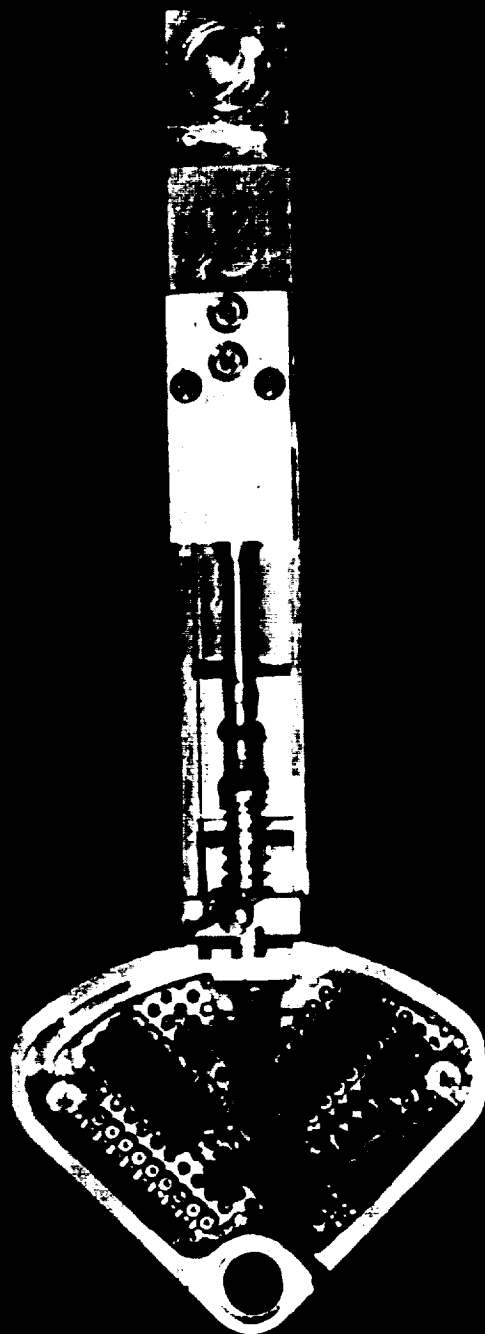


FIGURE 2.1-6

COMPLETED ARM ASSEMBLY



STAFF

FIGURE 2.1-7

addressed. Figure 2.2-1 shows the geometry of the field and the arm and disk angles. For each location in the field, there are two possible orientations of the arm and disk. If a low-Earth-orbit satellite passes directly through or very near the center of the field, there will be a short outage while the disks swing 180° to pick up the signal on the other side of center. These outages will be rare and are readily predicted well in advance.

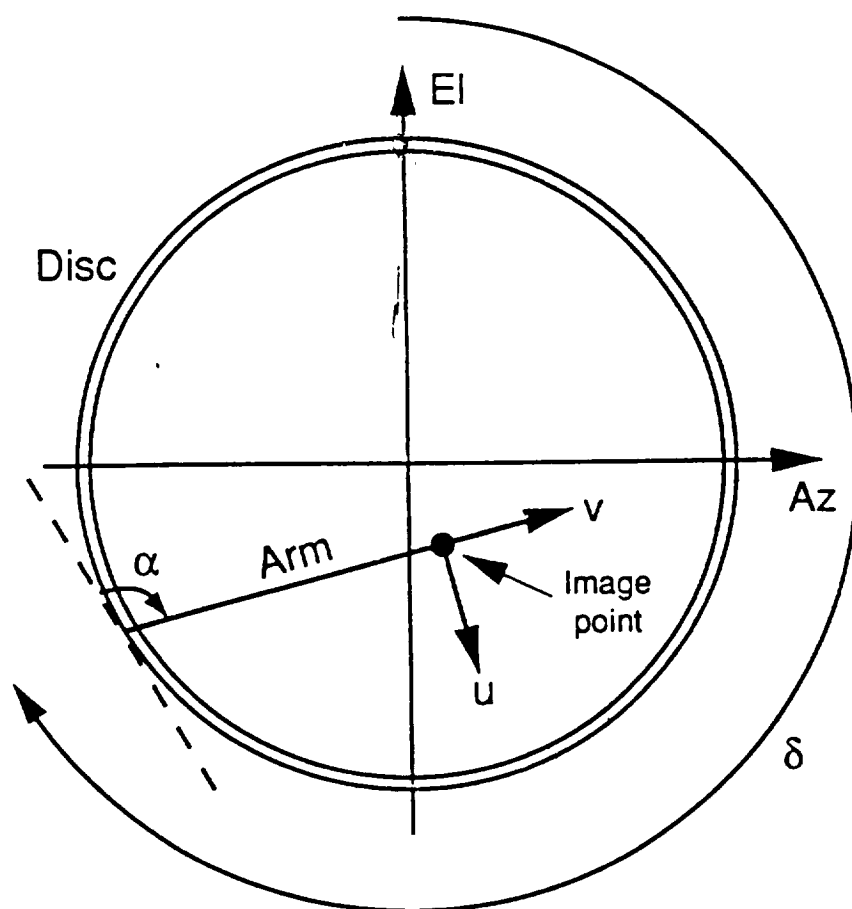
The worm assembly is shown in Figure 2.2-2. The worm is captive between preloaded bearings with no contribution to backlash. The assembly is spring-loaded into the worm gear to minimize backlash in the disk drive. The motors are of two types. Channel 1 is driven by two MicroMo 3557CR motors with up to 8.5 in-oz of continuous torque. The other two channels are driven by MicroMo 2842 motors with 3 in-oz of torque. The two different types of motors were used to ensure adequate torque margin on at least one channel, while exploring the possible savings in size and weight of the smaller motor. Position feedback for each motor is provided by a Hewlett-Packard HEDS 5010 2000-count-per-revolution incremental optical shaft encoder.

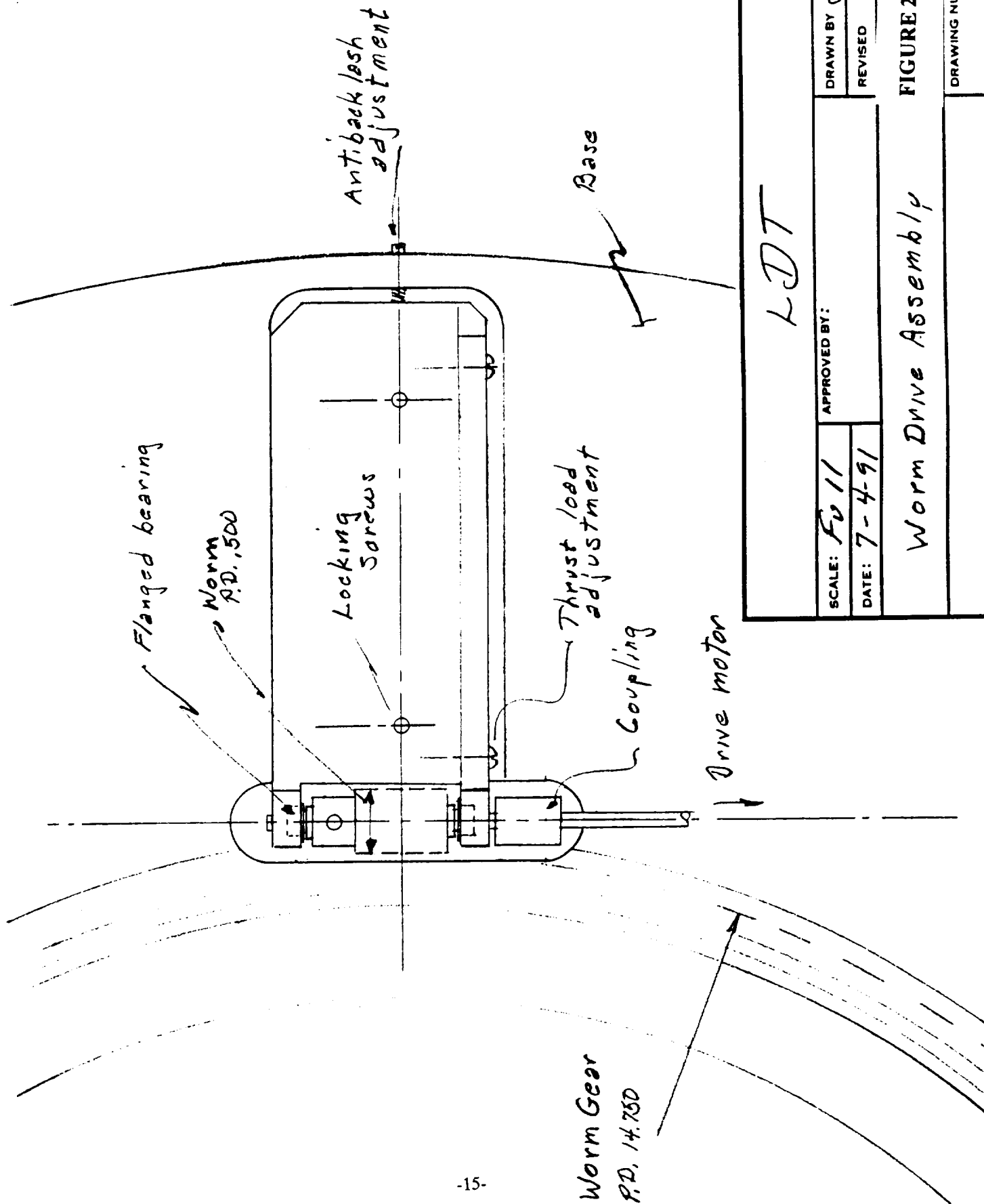
The 3-channel structure is shown in cross-section in Figure 2.2-3 showing the "sandwich" construction of the disk assembly. A photo showing the baseplate with the motor and worm interfaces is presented in Figure 2.2-4. Figure 2.2-5 gives the mechanical constants for the arm and disk mechanical design.

2.3 Acquisition and Tracking Design Constraints

The approach for acquisition is simple and straightforward. Spacecraft attitude reference, ephemeris, and mounting uncertainties require that a square 3.5 milliradians on a side be searched at the GEO. Table 2.3-1 presents the analysis of this requirement. The LEO will stare at its uncertainty region and respond as soon as it is illuminated by slewing its narrow transmit beam to reduce the error measured by means of the received signal from the GEO. Consequently, if the GEO will dwell at a given location for a time sufficient for the signal to reach the LEO (1 transit time), the LEO to slew to boresight (say, 30 milliseconds), and return its optical signal (1 transit time), then before it moved to the next scan location, it would know to stop where it is and track. Figure 2.3-1 shows the resulting worst case acquisition times given 100% probability of detection when illumination occurs.

FIGURE 2.2-1 ARM AND DISK COORDINATES IN THE TELESCOPE FIELD OF VIEW





APPROVED BY:

SCALE: Full

REVISED

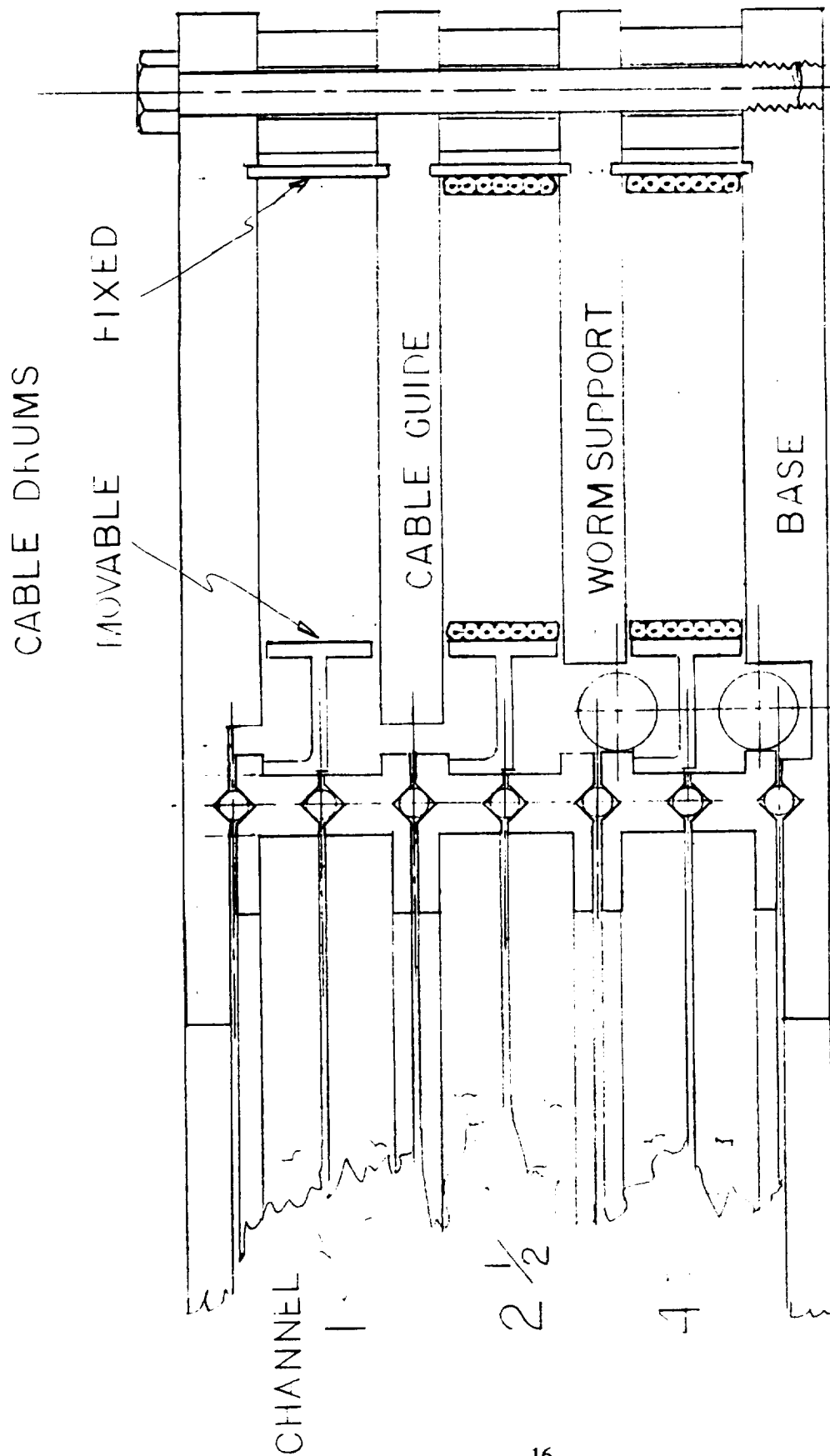
DATE: 7-4-91

DRAWN BY DWI

FIGURE 2.2-2

Worm Drive Assembly

DRAWING NUMBER



LIT

APPROVED BY:

SCALE: Full

DRAWN BY DM

REVISED

DATE: 7-2-91

DISK SUPPORT STRUCTURE FIGURE 2.2-3

DRAWING NUMBER

DISK ASSEMBLY END VIEW

FIGURE 2.2-4

DISK/ARM RESOLUTION

CONTINUOUS MOTOR/WORM DRIVE IMPLEMENTATION

GEAR TEETH PER REVOLUTION 24 PITCH ON 14.75-INCH DIAMETER	354
WORM DRIVE ENCODER RESOLUTION	2000
DISK STEP RESOLUTION $2\pi/354/2000 =$	$8.87 \mu\text{RAD}$
CORRESPONDING MOTION AT EDGE OF 116-MM FOCAL PLANE $8.87 \mu\text{RAD} \times 116\text{mm}$	$1.03 \mu\text{m}$
CORRESPONDING FAR-FIELD ANGLE $1.03 \mu\text{m}/.625\text{m}$ (FOCAL LENGTH)	$1.64 \mu\text{RAD}$

FOR D4/D3 = 2.56

ARM STEP ANGULAR RESOLUTION $8.87 \mu\text{RAD} \times 2.56$	$22.7 \mu\text{RAD}$
LINEAR MOTION AT TIP OF ARM $22.7 \mu\text{RAD} \times \text{ARM LENGTH (167mm)}$	$3.79 \mu\text{m}$
CORRESPONDING FAR-FIELD ANGLE $3.79 \mu\text{m}/.625\text{m}$	$6.07 \mu\text{RAD}$

FIGURE 2.2-5

ACQUISITION ERROR BUDGET

	STEL ATTITUDE MODEL	AZIMUTH	ELEVATION
GEO ATTITUDE KNOWLEDGE			
Pitch	± 1 mrad	± 1 mrad	0
Roll	± 1 mrad	0	± 1 mrad
Yaw	± 3 mrad	± 0.5 mrad	± 0.5 mrad
OTHER ERRORS	± 0.25 mrad	± 0.2 mrad	± 0.2 mrad
WORST CASE		± 1.7 mrad	± 1.7 mrad

TABLE 2.3-1

ACQUISITION APPROACH

- LEO STARES WITH RECEIVE IFOV GREATER THAN UNCERTAINTY WINDOW
- GEO SCANS WITH 200 μ RAD TRANSMIT BEAM AND 500 μ RAD IFOV COVERING UNCERTAINTY WINDOW WITH 50% OVERLAPS
- GEO DWELLS LONG ENOUGH AT EACH SPOT FOR LEO TO ACHIEVE FINE TRACKING IF WITHIN TRANSMIT BEAM
 - ROUND-TRIP DELAY
 - CONVERGENCE TIME
 - TOTAL DWELL TIME

0.27 SECONDS
0.03 SECONDS
0.30 SECONDS

INITIAL ACQUISITION TIME

UNCERTAINTY REGION
DWELLS
MAXIMUM ACQUISITION TIME

3.4 X 3.4 MRAD
1089
5.4 MINUTES

SUBSEQUENT ACQUISITION TIME

UNCERTAINTY REGION
DWELLS
MAXIMUM ACQUISITION TIME

1.2 X 1.2 MRAD
136
41 SECONDS

RE-ACQUISITION TIME

UNCERTAINTY REGION
DWELLS
MAXIMUM ACQUISITION TIME

<1 X 1 MRAD
<100
<30 SECONDS

FIGURE 2.3-1

2.4 Electronics

Figure 2.4-1 presents an overall block diagram of the electronics showing where the various circuits are housed. The computer contains the motor control electronics on a special internal expansion card that plugs into the computer bus. The power supplies, limit-switch relay circuits, and computer interface connectors are mounted in the cradle that supports the telescope and its focal-plane optics assemblies. The quadrant-detector receiver circuits are mounted in the arm assemblies themselves and connect to angle-processing circuitry mounted on the disk assemblies. Connection to the computer from each arm is made via the cradle-mounted circuitry through a flexible cable which permits $\pm 240^\circ$ travel of each arm.

Summary descriptions of the electronic assemblies are presented below. Detailed schematics of all circuits are contained in Appendix B.

2.4.1 Motion control circuitry - Figure 2.4-2 is a block diagram of the motion-control chip used for control of the motors driving the worm-gear assemblies. The performance of the compensation loop is controlled by downloading the desired gain, cycle time, and digital transfer function parameters.

The overall control loop is shown in Figure 2.4-3. This diagram incorporates the functions of the HCTL-1000 motor-control chip, the motor drive amplifier, the quadrant detector and its angle processor, and the computer control algorithms for open- and closed-loop control of the arm position for each channel. The design of the control loop sought to achieve a 4-Hertz control bandwidth, supported by a 40-Hertz update rate for each channel by the computer. Figure 2.4-4 shows the design Bode plots for the open- and closed-loop motor control transfer functions for a 4-Hertz control loop.

Motion control circuit board - This board contains six Hewlett-Packard HCTL-1000 chips and the associated interface circuitry. This chip accepts input position commands, incremental encoder outputs, and mode-control and loop-compensation parameters and develops the appropriate motor drive word for D/A conversion and amplification. Other circuitry performs multiplexing of the computer bus among the various motor-control chips and output word conversion.

SYSTEM ELECTRONICS LAYOUT

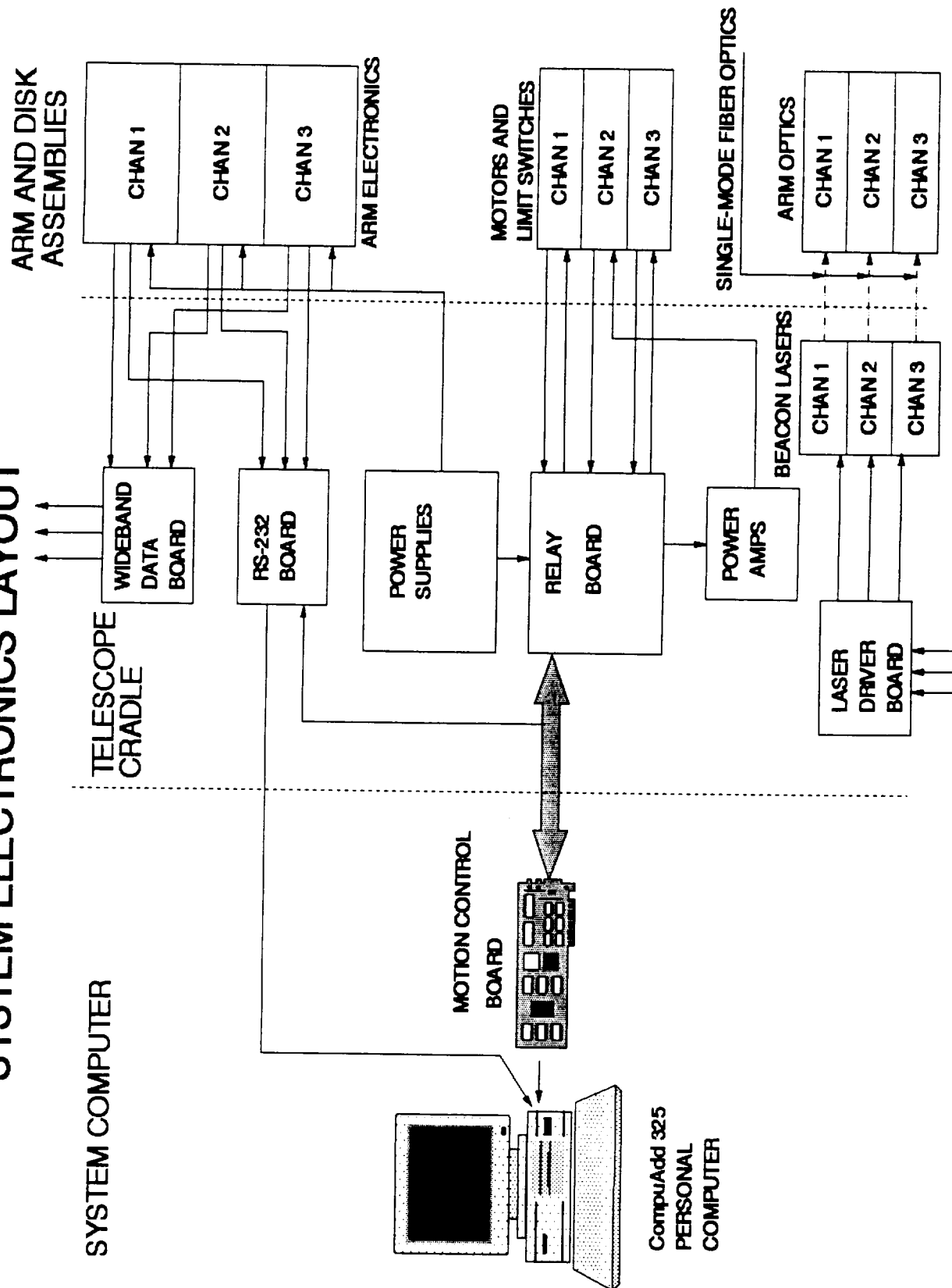


FIGURE 2.4-1

MOTOR CONTROLLER CHIP BLOCK DIAGRAM

HEWLETT-PACKARD HCTL-1100

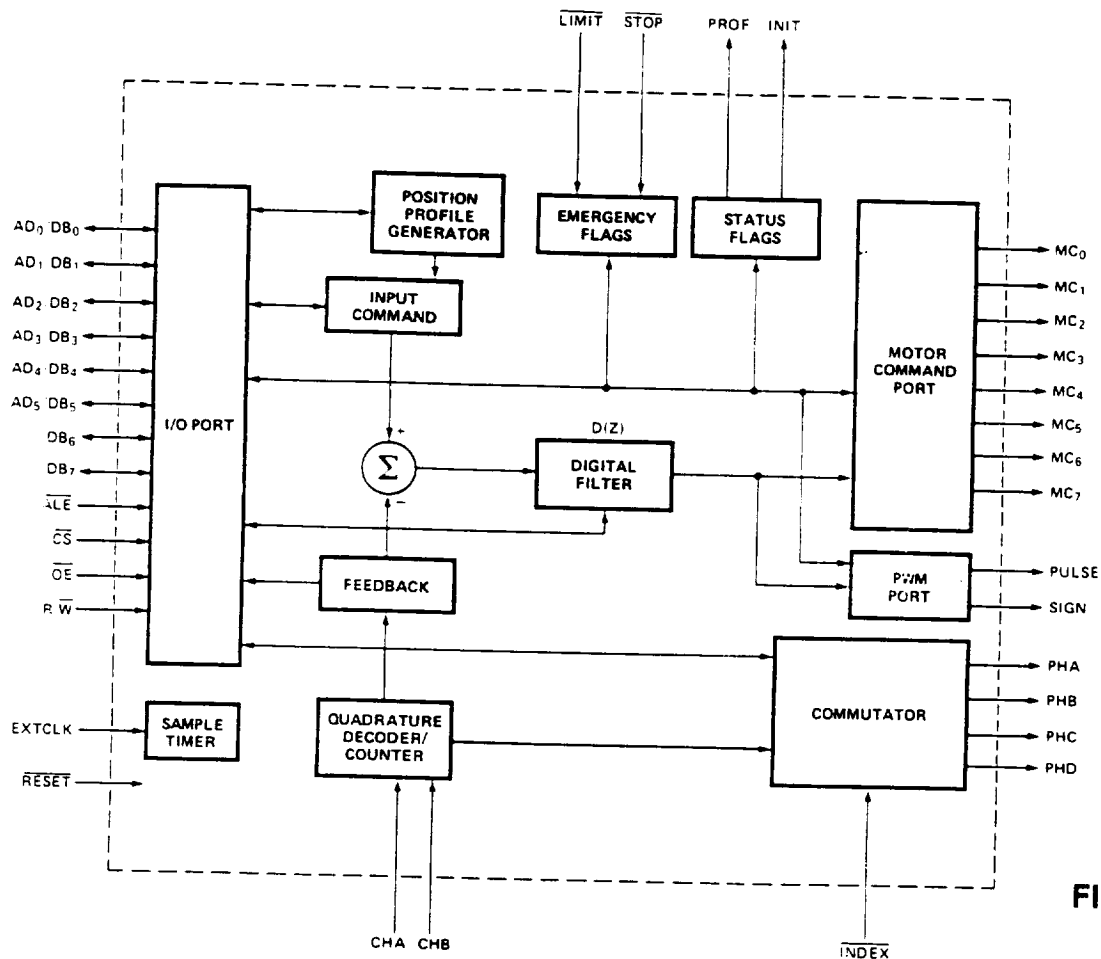


FIGURE 2.4-2

TOTAL CONTROL LOOP

AZIMUTH, ELEVATION ANGLES

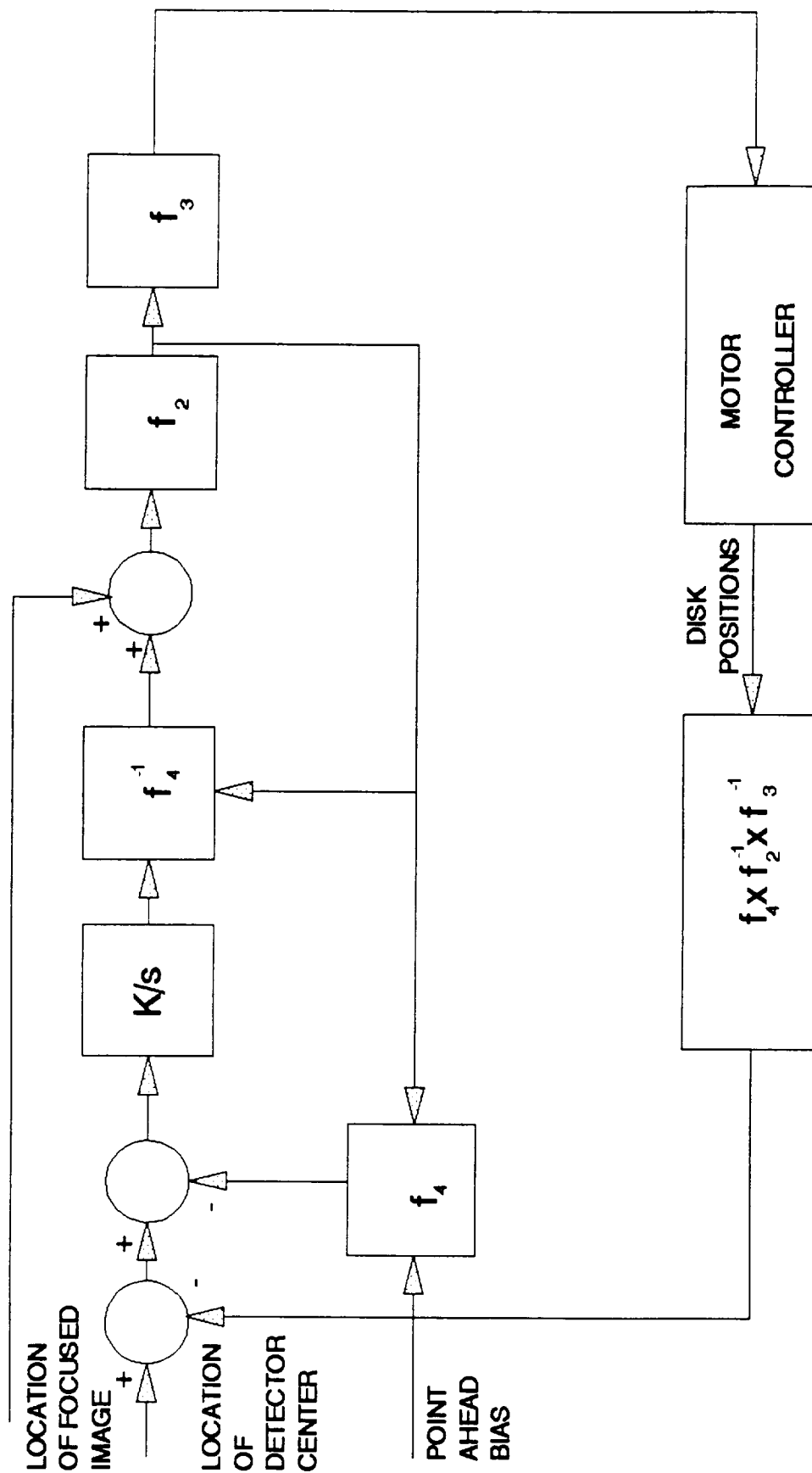


FIGURE 2.4-3

MOTOR AND TOTAL LOOP RESPONSES

-Voltage Control Mode-

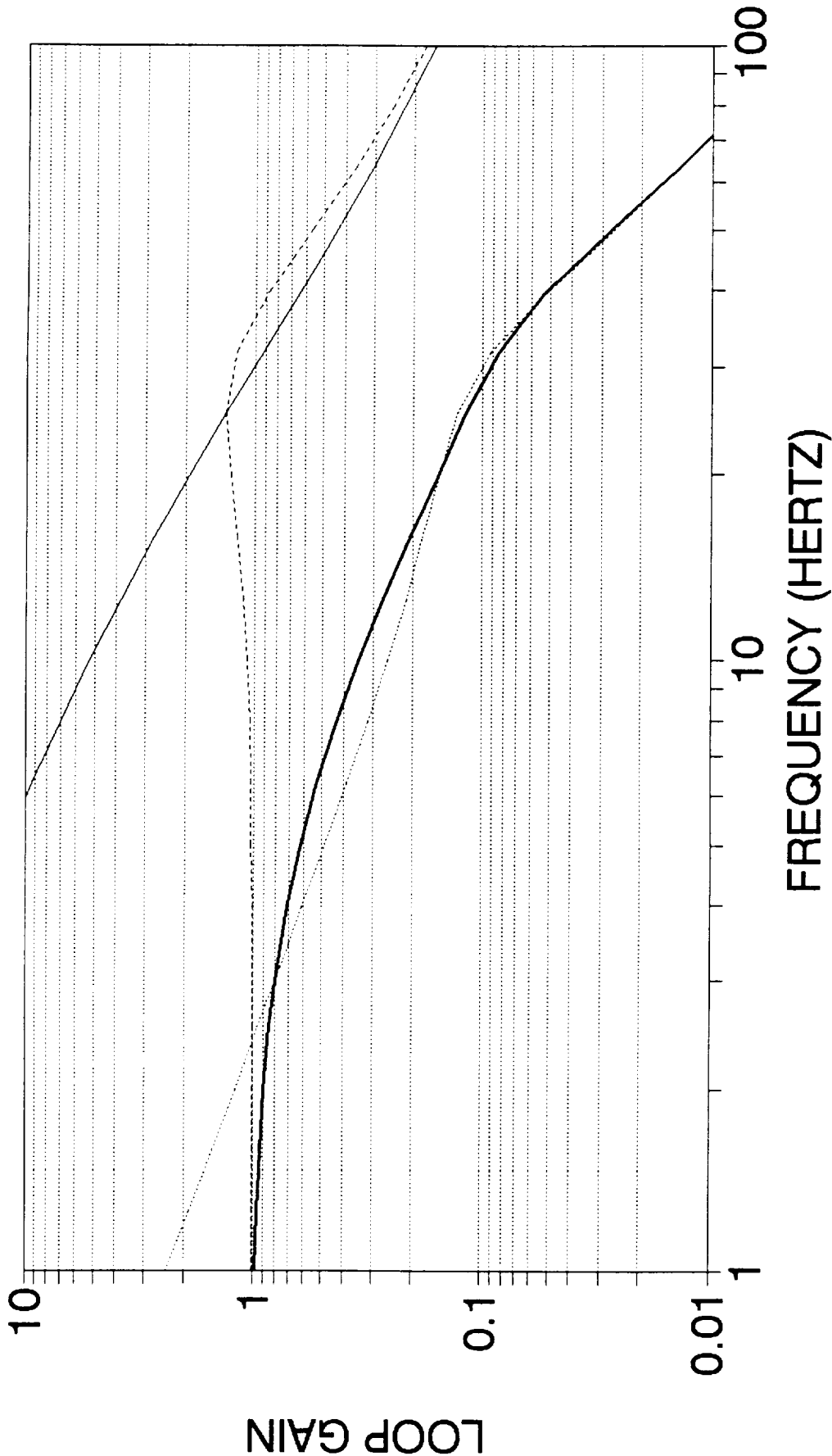


FIGURE 2.4-4

— Motor OLG - - - Motor CLG . . . Total OLG — Total CLG

Limit-switch relay circuit board - Each of the three channels has four limit switches to sense the positive and negative limits of motion for the disk pair and for the arm. These switches operate relays which prevent current flow in the appropriate motor in the appropriate direction to prevent further motion in the offending direction but which permit motion in the opposite direction. Encoder outputs and motor drive signals are routed through this board, which is mounted in the cradle assembly. Figure 2.4-5 shows the arrangement for Channel 2. Channels 1 and 3 are similar except that the power amplifiers are not required for the smaller motors used on those channels.

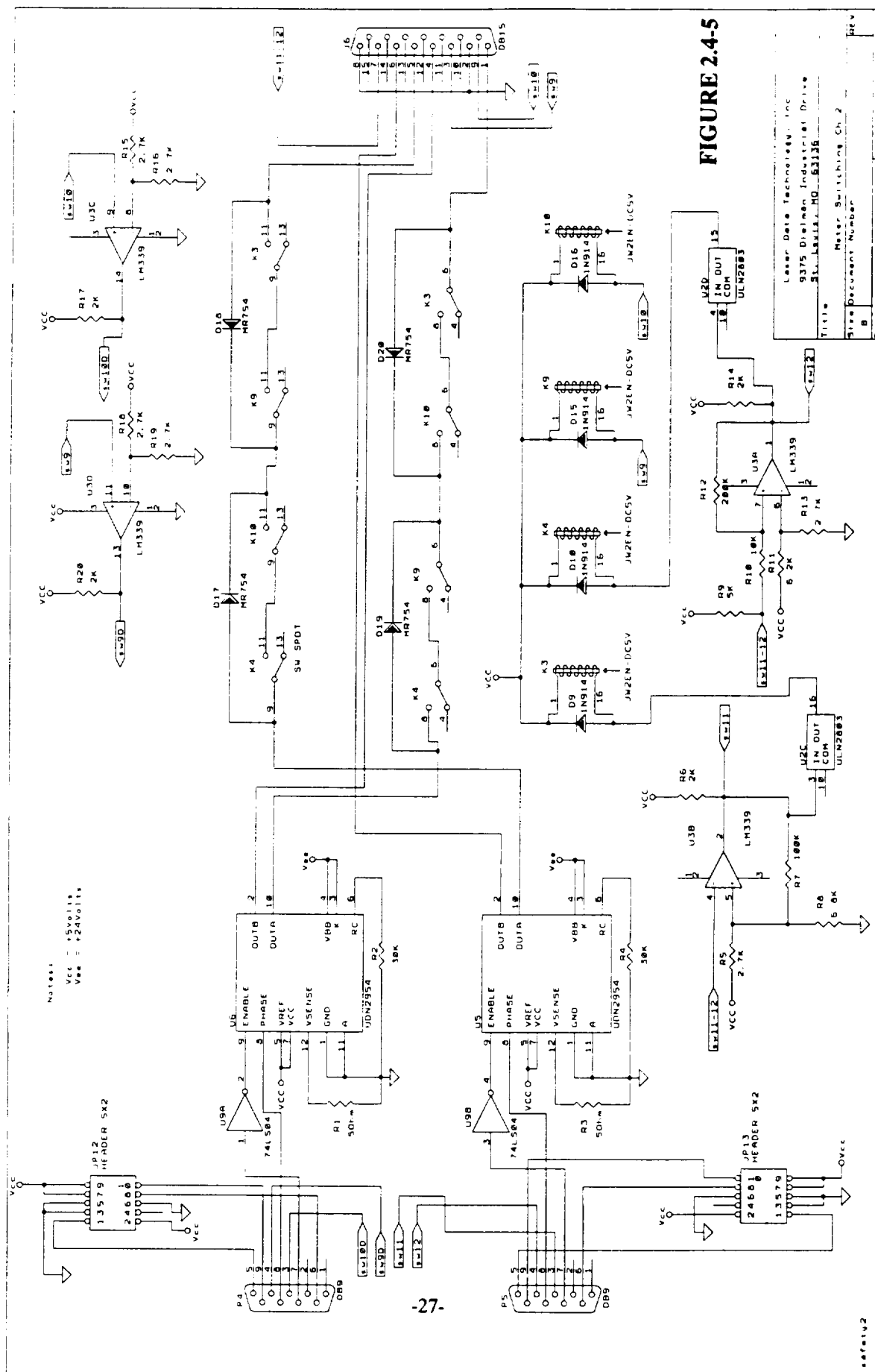
Quadrant detector preamplifier - Two stages of preamplification are utilized to detect a 5 kHz tone on the received optical signal incident on the four segments of the quadrant detector. The stages are AC coupled and rolled off above 6 kHz to eliminate DC drifts and high frequency components in of the communications data signal. A low-noise operational preamplifier, OP-470, and post amplifier, OP-471, from PMI are used for this purpose. See Figure 2.4-6.

Digital angle processor - The Analog Devices ADSP-2101 is a digital signal processor utilized to process samples of the four quadrants to produce a measure of the position of the centroid of the optical signal focused on the quadrant detector. The familiar horizontal and vertical difference-over-sum functions of the four quadrant signals, A, B, C, and D, are given by

$$\delta_v = (A+B-C-D)/(A+B+C+D)$$

$$\delta_h = (A+C-B-D)/(A+B+C+D)$$

The processor computes the numerators of the above expressions in addition to the more demanding task of extracting the amplitude of the 5-kHz tone from the composite data and tone signal. Figure 2.4-7 illustrates the signal waveform. The amplitude of the basic Manchester pulse train is modulated with the 5-kHz tone to a depth of 10%. The preamps described above perform some initial processing by attenuating the components at the data frequency. The resulting 5-kHz signal is sampled at 27.18 kHz by a 4-channel A-to-D converter which simultaneously samples and holds the 4 quadrant signals then sequentially converts them for use by the processor. The processor passes 100 such samples for each quadrant through a 40-tap bandpass



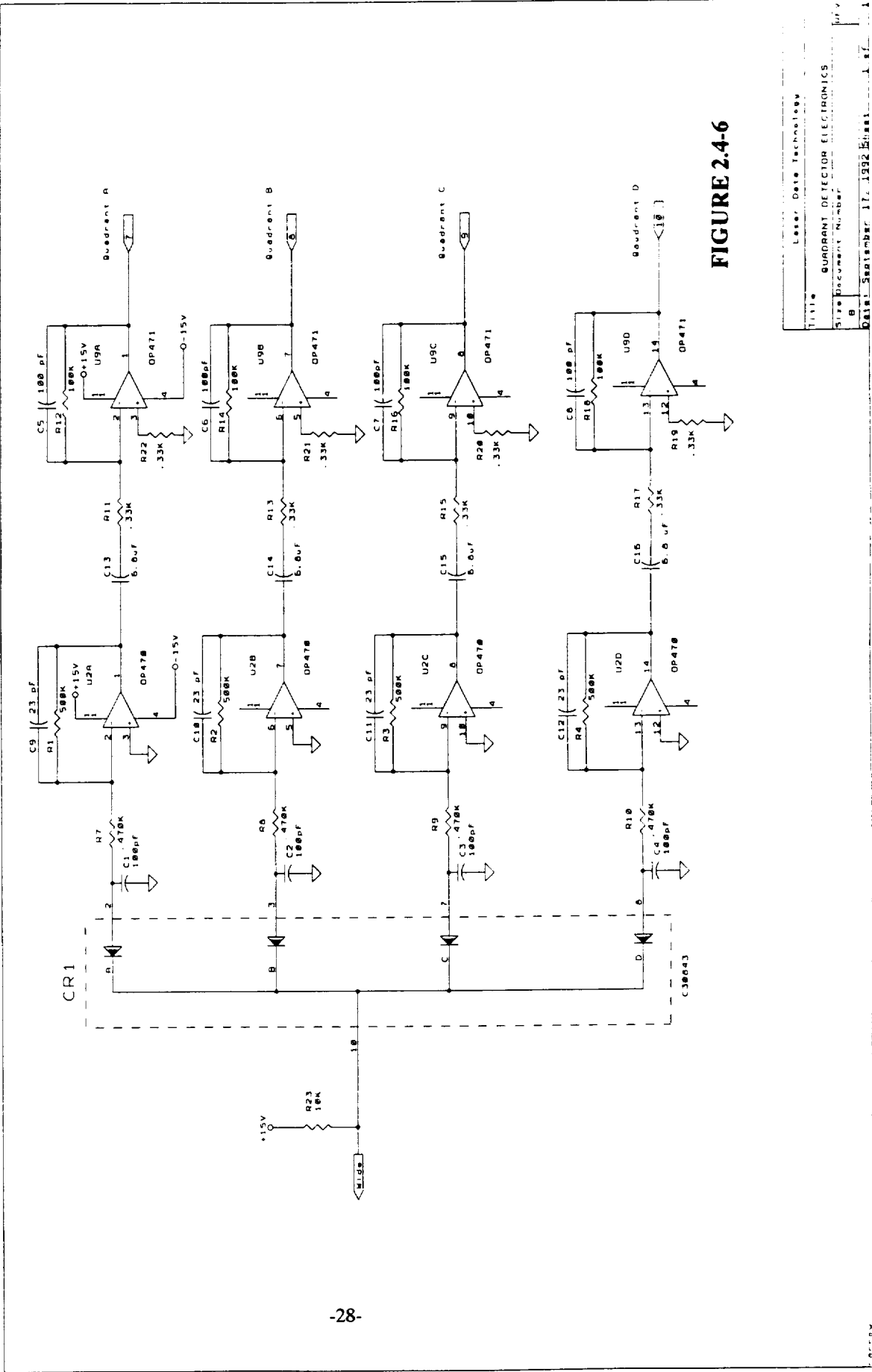


FIGURE 2.4-6

Laser Data Technology	
Title	QUADRANT DETECTOR ELECTRONICS
Size	10.0
Sheet	1 of 1
Date	September 17, 1992

RECEIVER WAVEFORM SHOWING DATA AND TONE

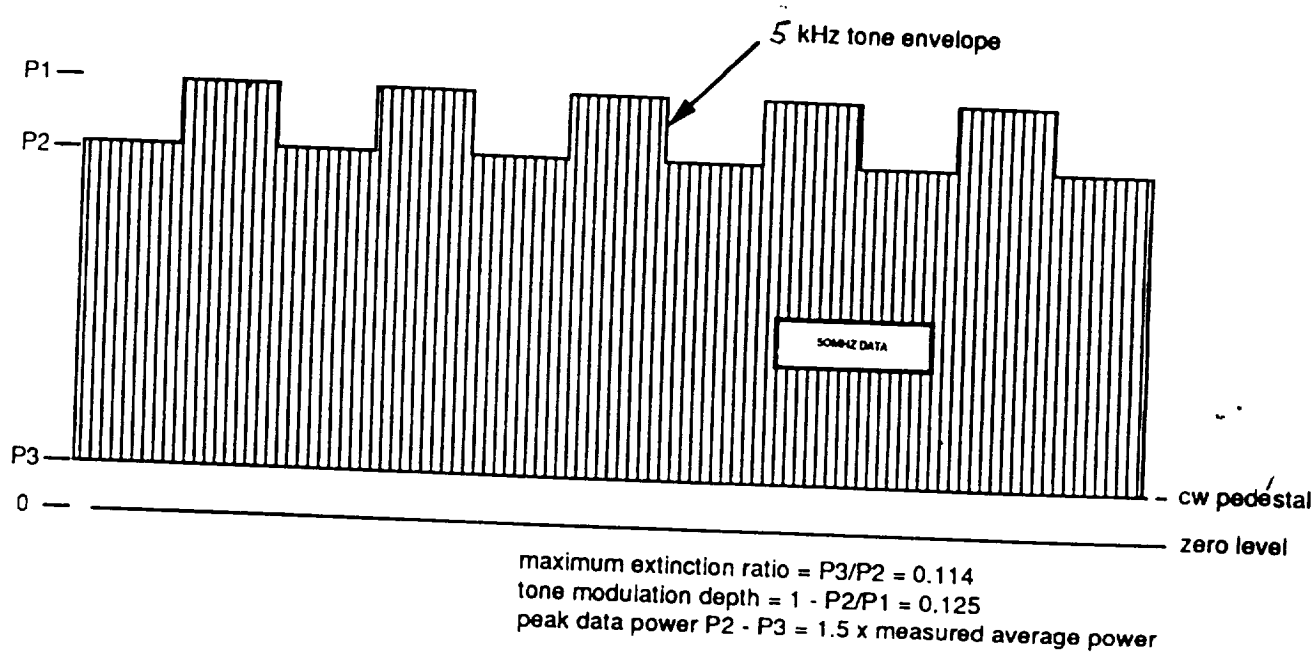


FIGURE 2.4-7

digital filter, a rectifier, and a 40-tap low-pass to extract the amplitude of the tone. These four values are then used in the above equations to produce the vertical and horizontal position components. The entire process requires 4.2 milliseconds and has a latency of about 7.8 milliseconds, i.e., the data used to control the motors is about 7.8 milliseconds old when the resulting control is applied. The circuit configuration is presented in Figure 2.4-8. The software for this processor is described in the software section.

2.4.2 Communications circuits - The quadrant detector bias current is modulated by the total optical signal incident on the detector and contains the wideband data signal. Figure 2.4-9 shows the wideband amplifier used to detect the data component of the optical signal. The TIEF151 amplifiers have 40-MHz bandwidth and a transimpedance gain of 4 kilohm. The arrangement utilized here achieves a closed-loop bandwidth 3 MHz to support the 3 Mbps receive-channel design data rate. Higher bandwidth is achievable with the detector used given the required amplifier design and packaging developments.

The transmitted optical signal serves as a tracking beacon to the low-Earth-orbit (LEO) satellite and provides low-rate data capability for command, control, and housekeeping functions. Figure 2.4-10 presents the circuit to drive an SDL-5301 laser diode to impose a 5-kHz. tone and 100-kHz. data signal on the optical beam. Adjustments for the diode slope efficiency and threshold characteristics to achieve the desired depth of modulation for the tone are provided. These circuits are mounted in the cradle area with the laser diodes they drive, and the optical signal is fed to the arm optics in each case by means of a single-mode fiber.

2.5 Software

Executive control of the terminal is provided by a CompuAdd 325 personal computer with a math co-processor. The motor control processor described above receives its operating parameters and mode control instructions from the executive while the angle processors, triggered by a sync-pulse generator, acquire the latest tracking errors. Overall synchronization of the executive, motor-control, and angle processors is accomplished by the sync-pulse generator. This software is described in this section. Detailed listings with comments are contained in Appendix C.

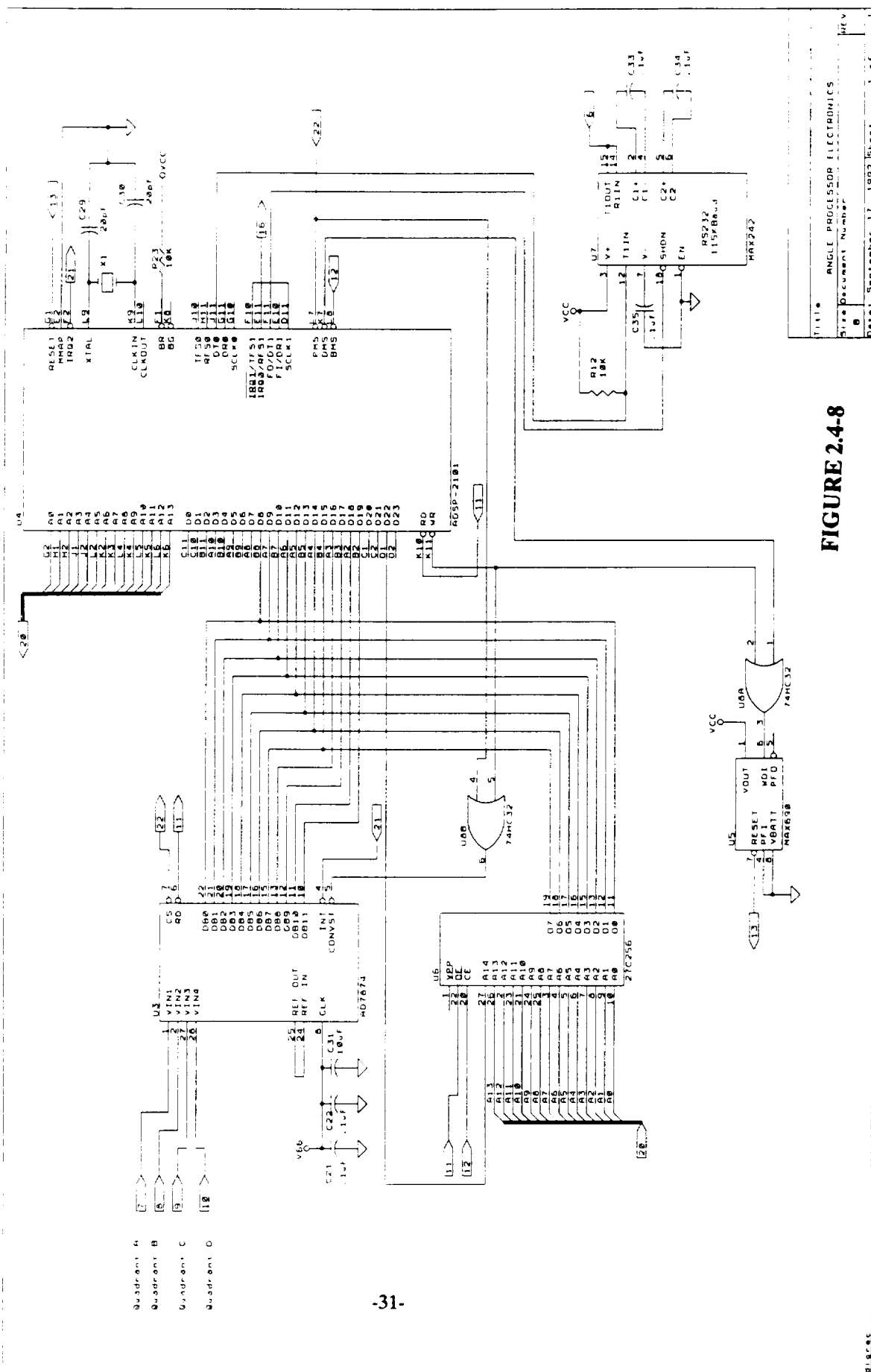


FIGURE 2.4-8

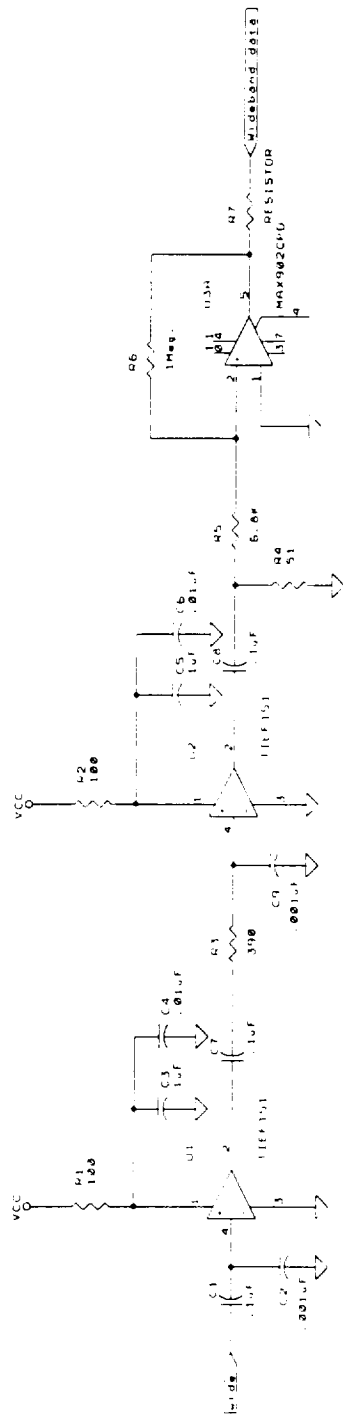
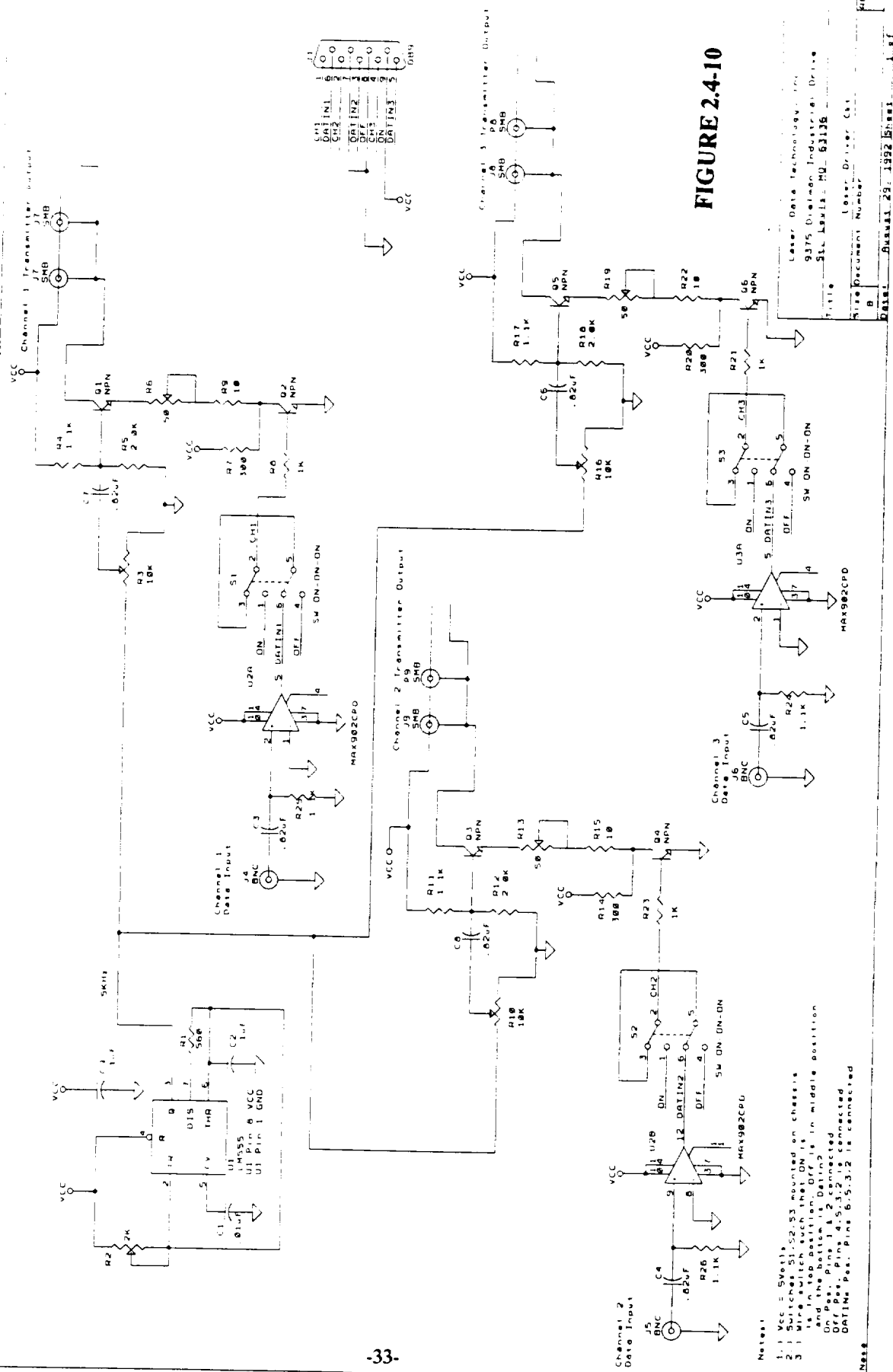


FIGURE 2.4-9

Laser Data Technology, Inc.	
9375 Oldham Industrial Drive	
St. Louis, MO 63132	
Title	
Wideband Amplifier	
Size Document Number	
0	
Date: August 29, 1992	
Sheet 1 of 1	



2.5.1 Executive - There are three main executive programs for operation of the terminal: MANUAL.C, TRACK.C, and OMA.FOR. The first two programs, written using Microsoft Quick C are used for real-time control of the terminal when operated in the manual or automatic mode, respectively. In addition, there is a Fortran executive, OMA.FOR, which calls the other Fortran programs to exercise a simulation of the system, and which represents an initial version of satellite operational executive software. The Fortran and C programs are compiled into MS-DOS-executable files to perform the complete job of operating the terminal in the modes required.

Manual Operation Software - Table 2.5-1 illustrates the make-up of the manual MS-DOS-executable file, MANUAL.EXE. As seen from the table, these programs are written entirely in C. Their functions are accessed from the main menu, presented in Table 2.5-2. These functions permit open-loop exercising of the disk and arm actuators for the three channels. For example, initial calibration of the position references for all channels can be accomplished with Option 6, Channel Alignment. This action is required only if a disk has been moved by hand either with the power off or with neither MANUAL.EXE nor TRACK.EXE running. Arm and disk motion to specific locations can be accomplished with Options 1 and 2. Exercising channels to troubleshoot or validate mechanical operation can be accomplished with Options 3 and 4. Various tests of the motion-control electronics can be performed with Options 7-14.

Operational Satellite Software - The operational satellite software is illustrated by Figure 2.5-1, which diagrams the interaction of the various Fortran modules of the control software for an operational system. Where arrows indicate hardware interfaces, this software simulates the hardware when necessary to complete an action in the simulation mode. This organization was developed during the design phase of this project in order to understand the constraints imposed by the satellite and the mission. The modifications required to interface with and operate the prototype terminal preserve this organization to the maximum extent possible. They are discussed in Paragraph 2.5.4.

Table 2.5-3 illustrates the make-up of OMA.EXE from source code files written in Fortran. Each terminal interface module is sensitive to a REALTIME discrete to determine whether it simulates the inputs it needs or gets them from a hardware source. Therefore the programs are useful for simulation studies and also contain the switches to permit interfacing with actual hardware.

TABLE 2.5-1 MANUAL OPERATION SOFTWARE ORGANIZATION

Modules called by MANUAL.C to make up MANUAL.EXE

MODULE FILE	FUNCTION
COMPUTE.C	Subroutines to convert among az/el, alpha/delta, and delta A/B coordinates.
DISCRETE.C*	Subroutines to monitor the status of all the discretes in the system.
DISPLAY.C	Subroutines to generate the menu for the manual operation.
FILE.C*	Subroutines to open and close all files, read and write system and test data.
INITMOT.C	Initializes motion control digital parameters.
MOTCNT.C*	Subroutines for mechanical alignment of the system.
MOTOR.C*	Subroutines for direct interfacing with the motor-control circuit board in the computer.
NEWMOT.C	Subroutines for controlling the motion of the disks.
SERL.C*	Subroutines for communication over RS-232 links from the PC to the channel angle processors or to another PC.

*Also used in the automatic mode presented in Table 2.5-3.

TABLE 2.5-2 MANUAL CONTROL MENU

*** Move Channel Commands ***

1. Move optics arm delta & alpha increment
2. Move arm to designated Az and El
3. Exercise single channel through limits
4. Exercise all channels through limits
5. Compute channel Az & El

*** Alignment Command ***

6. Channel alignment

*** System Test Commands ***

7. Motor safety switch test
8. DSP to PC serial link test
9. Host PC to Monitoring PC serial link test
10. Power monitoring discrete test
11. Motor drive backlash test
12. Motor step response test
13. Channel eccentricity test
14. Reset Motor Command

CONTROL SOFTWARE TOP-LEVEL ARCHITECTURE

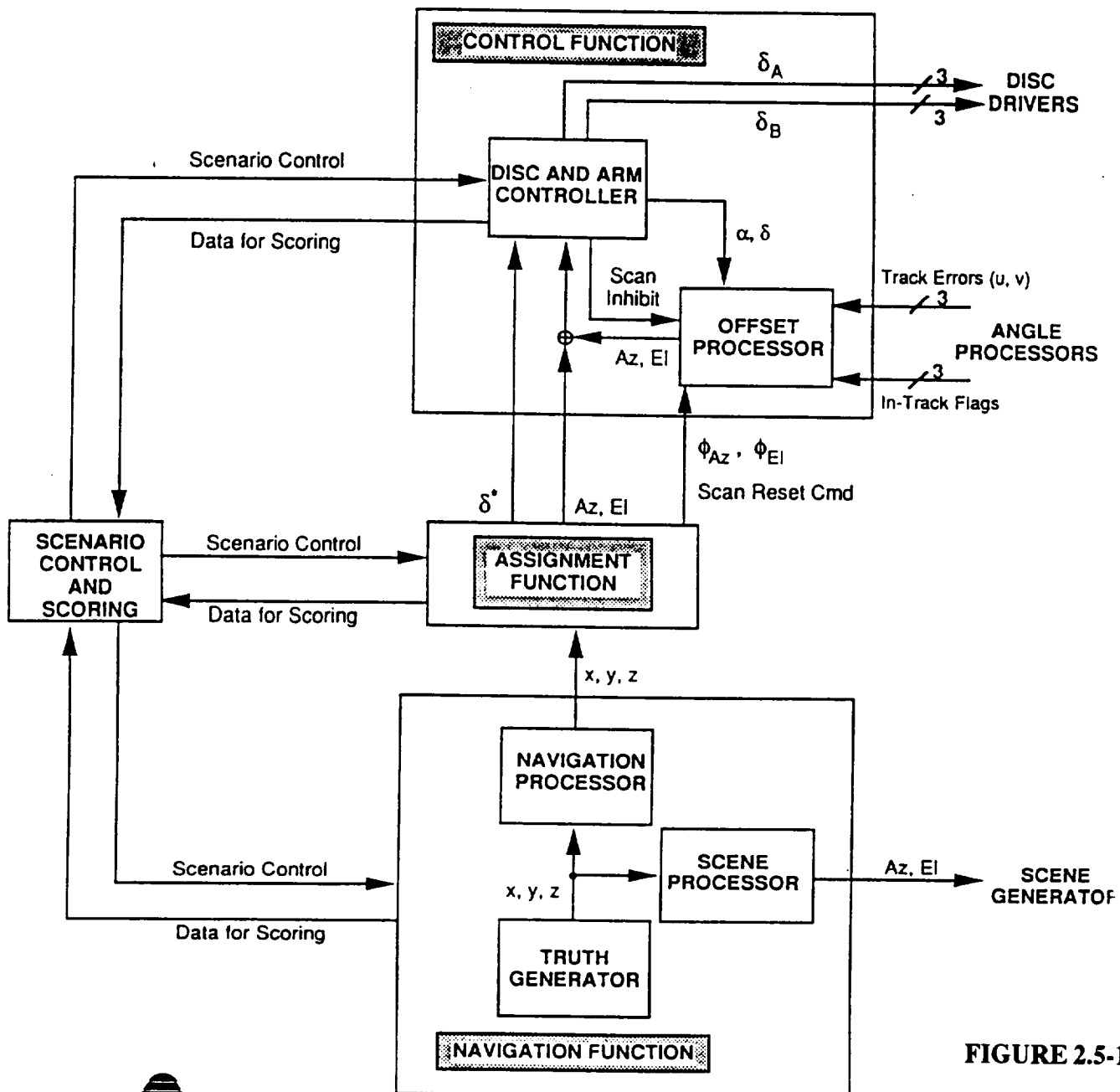


FIGURE 2.5-1



Avtec Systems, Inc.

AVD 90-0263-02

TABLE 2.5-3 OPERATIONAL SATELLITE SOFTWARE ORGANIZATION

Modules called by OMA.FOR or by each other to make up OMA.EXE

MODULE FILE	CALLED BY	FUNCTION
ALDLDLAB.FOR	CNTRLFN.FOR	Converts alpha/delta to disk A/B coord.
ANGLPR.FOR	CNTRLFN.FOR	Simulates angle processor in sim. mode.
ARKTNS.FOR	ANGLEPR.FOR	Takes arctan of arguments in 4 quadrants.
AZELALDL.FOR	ASGNMFN.FOR	Converts az/el to alpha/delta coordinates.
CNTRLFN.FOR	ASGNMFN.FOR	Provides the arm/disk position commands.
COPYVEC.FOR	TRUTHGEN.FOR	Copies a vector to a new name.
CROSS.FOR	XYAZEL.FOR	Performs vector cross product mathematics.
DFQ.FOR	RUK.FOR	Satellite state differential equations
DOT.FOR	ANGLEPR.FOR	Performs vector dot product mathematics.
ELSTAT.FOR	OMA.FOR	Converts orbital to Cartesian coordinates
GAUSCL.FOR	ANGLEPR.FOR	Generates random numbers from clipped Gaussian distribution.
GAUSS.FOR	GAUSCL.FOR	Generates random numbers from Gaussian distribution.
INV3X3.FOR	QFITVAL.FOR	Inverts a 3x3 matrix
NAVIGFN.FOR	OMA.FOR	Performs the navigation function
NAVIGPR.FOR	NAVIGFN.FOR	Adds ephemeris errors to true pos. and vel.
QFITVAL	TRUTHGEN.FOR	Quadratic time fit for pos. and vel.
RNDM.FOR	SCENEPR.FOR	Generates random numbers for noise sim.
RUK.FOR	TRUTHGEN.FOR	Fourth order Runge-Kutta integrator.
SCANGN.FOR	CNTRLFN.FOR	Scan generator for acquisition/reacquisition functions
SCENEPR.FOR	TRUTHGEN.FOR	Computes az/el for each LEO beam.
TRUTHGN.FOR	NAVIGFN.FOR	Simulates satellite orbital dynamics
UNIT.FOR	XYAZEL.FOR	Converts vector to unit vector.
VLEN.FOR	UNIT.FOR	Computes the length of a vector.
XKEP.FOR	ELSTAT.FOR	Solves Kepler's orbital equation.
XYAZEL.FOR	SCENEPR.FOR	Converts from Cartesian to OMA FOV coord.
	ASGNMFN.FOR	

Real-time Automatic Operation Software - The software of the previous section was modified to permit operation with the prototype hardware in a laboratory environment. The satellite trajectory information is replaced with an initial point for acquisition of a signal to be tracked. No effort was made to produce a trajectory in the test equipment that could be acquired by open-loop pointing at any point in the trajectory similar to acquiring a satellite. The sophistication, complexity, and cost of such elaborate test equipment was beyond the scope of this effort. Instead, the acquisition program scans a region known to intersect the trajectory of the test source so that when the source approaches it can be acquired. Table 2.5-4 illustrates the make-up of TRACK.EXE from source code files written in both C and Fortran. The Fortran portion controls the automatic acquisition and tracking functions while the C portion sets up the initial conditions and handles all system-level interfaces.

TABLE 2.5-4 AUTOMATIC REAL-TIME OPERATION SOFTWARE ORGANIZATION
Modules called by TRACK.C or by each other to make up TRACK.EXE

MODULE FILE	CALLED BY	FUNCTION
MENU.C	Subroutines to generate the menu for the automatic operation.	
Other C files	See Table 2.5-1	See Table 2.5-1
ARKTNS.FOR	ANGLEPR.FOR	Takes Arctangent of arguments in 4 quadrants.
AZELALDL.FOR	ASGNMFNR	Converts az/el to alpha/delta coordinates.
ANGLPR.FOR	CNTRLFN.FOR	Accesses real-time angle processor output.
ASGNMFN.FOR	INIT	Selects the channel and solution for assignment to a LEO beam
ASGNMFNR (sub)	Compiler	Global constants collected in one file.
BLKDATA.FOR	ASGNMFN.FOR	Provides the arm/disk position commands.
CNTRLFN.FOR	CNTRLFN.FOR	Accesses disk positions saved in C programs.
DISCPOS.FOR		
DOT.FOR	ANGLEPR.FOR	Performs vector dot product mathematics.
GAUSCL.FOR	ANGLEPR.FOR	Generates random numbers from clipped Gaussian distribution.
GAUSS.FOR	GAUSCL.FOR	Generates random numbers from clipped Gaussian distribution.
INTRFACE.FOR	TRACK.C	Interface between executive and Fortran.
INIT (sub)		Initializes control software.
POSITION (sub)		Computes new disk position commands.
SCANGN.FOR	CNTRLFN.FOR	Scan generator for acquisition/reacquisition functions

2.5.2 Angle Processor Software - The Analog Devices ADSP-2101 digital signal processor was programmed to extract the 5-kHz components from each of the 4 quadrant signals, then produce the horizontal and vertical coordinates of the centroid of the spot by sums and differences as described in Paragraph 2.4.1. The system was set up to sample each quadrant at 27.18 kHz for 100 samples and process these samples in an algorithm described by Figure 2.5-2. The files which make up this set of algorithms are listed in Table 2.5-5.

TABLE 2.5-5 ANGLE PROCESSOR ROUTINES

MODULE FILE	CALLED BY	FUNCTION
ADASYS.DSP	MAIN.DSP	Contains all routines for sampling data.
ALGORITHM.DSP	MAIN.DSP	Computes the un-normalized tracking error.
BANDFILT.DSP	ALGORITHM.DSP	Code for bandpass filter in Figure 2.5-2.
LOWPASS.DSP	ALGORITHM.DSP	Code for low-pass filter in Figure 2.5-2.
MAIN.DSP	On power-up	Executive software for angle processor.
UART.DSP	MAIN.DSP	Routine for formatting and sending data to host PC.

2.5.3 Motion Processor Software - The HP HCTL-1000 has built-in software to implement the functions illustrated in Figure 2.4-2. The executive is programmed to load the appropriate parameter values into the processor to effect the desired digital filtering characteristic and controls the modes of the device in the real-time operation of the system.

ERROR SIGNAL AMPLITUDE DETECTION

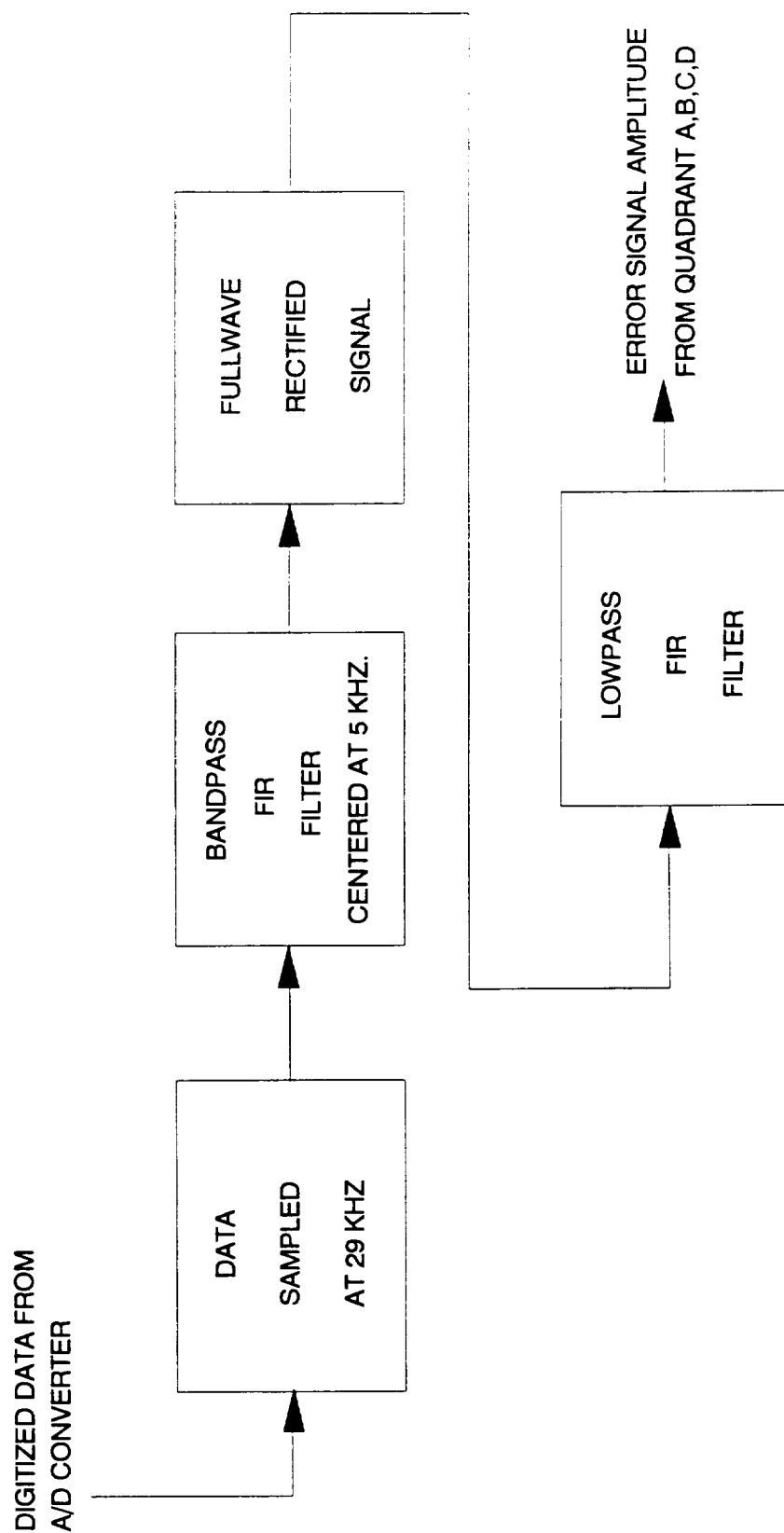


FIGURE 2.5-2

3.0 EVALUATION TEST DATA

Prior to delivery of the system to NASA Goddard, as part of the integration process we performed some evaluation tests on the system and its components. More extensive testing could guide the continuing development of an operational terminal by providing concrete results to design improvements economically before applying them to the next-generation hardware.

3.1 Optical Tests

3.1.1 Telescope Performance - The telescope transmission was evaluated using a HeNe laser at 6328 Å to be 67%. There are 4 lenses with two surfaces each for a total of 8 surfaces each apparently transmitting about 95% of the incident energy. The performance at the operational wavelength should have been better since the AR coating would be superior nearer the operational wavelength. The space system should be capable of significantly better performance with superior space coatings.

It was noted that near but not at the focus, the spot shape to the naked eye had a triangular shape that suggested distortion of the lens shape by the 6-point mounting hardware. We determined which lens of the set of four was misshapen by rotating each lens with respect to the remaining three and observing the orientation of the triangle before and after each such rotation. The spot rotated with the rotation of the frontmost element, i.e., the lens furthest from the focal plane, suggesting that it alone was distorted. We conclude from this test that (1) the mounting rings for both the front pair and the back pair should have more fasteners around each ring to equalize the clamping around the periphery of the lens, and (2) the interface between the two lenses in each pair should be a part machined to the correct shape rather than a flexible O-ring. This modification prevents the pair to be clamped in a distorted manner. It is possible but unlikely that the part was distorted in manufacturing. The nature of the distortion is not characteristic of manufacturing errors, which would be axisymmetric rather than triangular. The spot shape suggests a lens bent by externally applied force either from mishandling or asymmetrical clamping in the mounting bracketry. Because the ring had 6 equally spaced clamping bolts, it is reasonable that it could cause a distortion with three-fold symmetry, so as to produce a triangular spot. This aberration did not severely affect the focussed spot size based on the observations of spot shape on the detector given in

the next section.

3.1.2 Arm Performance - The arm optics were tested for transmission and for quadrant detector position transfer function. The transmissions were measured prior to assembly into the disk assembly at 65%, 61%, and 55% for Channels 1, 2, and 3, respectively. Figure 3.1-1 & -2 shows the quadrant processing characteristic for motion along one axis for Channel 3. These data were obtained prior to installation of the arm in the system. Note the S-shaped characteristic in the horizontal axis and the slight tilt in the vertical axis. This result shows a slight rotation of the detector about the optic axis. This error was corrected prior to installation and is presented to show the value of these tests.

The spot shape on the detector was inferred from scans of the arm past the test sources. Figure 3.1-3 shows a typical such scan using Channel 1 and diode #2 on a three-source test fixture. The derivative of the power on the detector versus position is a measure of the shape of the spot in the direction of the scan. Note that the focussed spot is twin-lobed due to the laser's characteristics but that the resolving power of the optical system is on the order of 100 microns.

3.2 Mechanical Tests

The mechanisms for the disks were tested and run in during assembly to remove roughness and minor imperfections in order to ensure smooth operation. Upon completion of assembly, tests for resolution, backlash, hysteresis and slew rate were performed. Tests to determine that the lighter motors are adequate to drive the worm assemblies when sufficiently preloaded to eliminate backlash were performed.

3.2.1 Resolution/backlash/hysteresis - Figures 3.2-1 and -2 show the role of the worm assembly adjustment mechanism of Figure 2.2-2. The "Displacement" and "Command" refer to the rotation of a single disk. Initial tests showed that the path traversed by the mechanism was very different depending on the direction of travel. After adjusting the worm into the worm gear, the hysteresis was significantly reduced and the current in the motors increased from .25 amps to .5 amps - still well within capacity. In an operational design, improved design and machining approaches would be considered to reduce friction and backlash, including

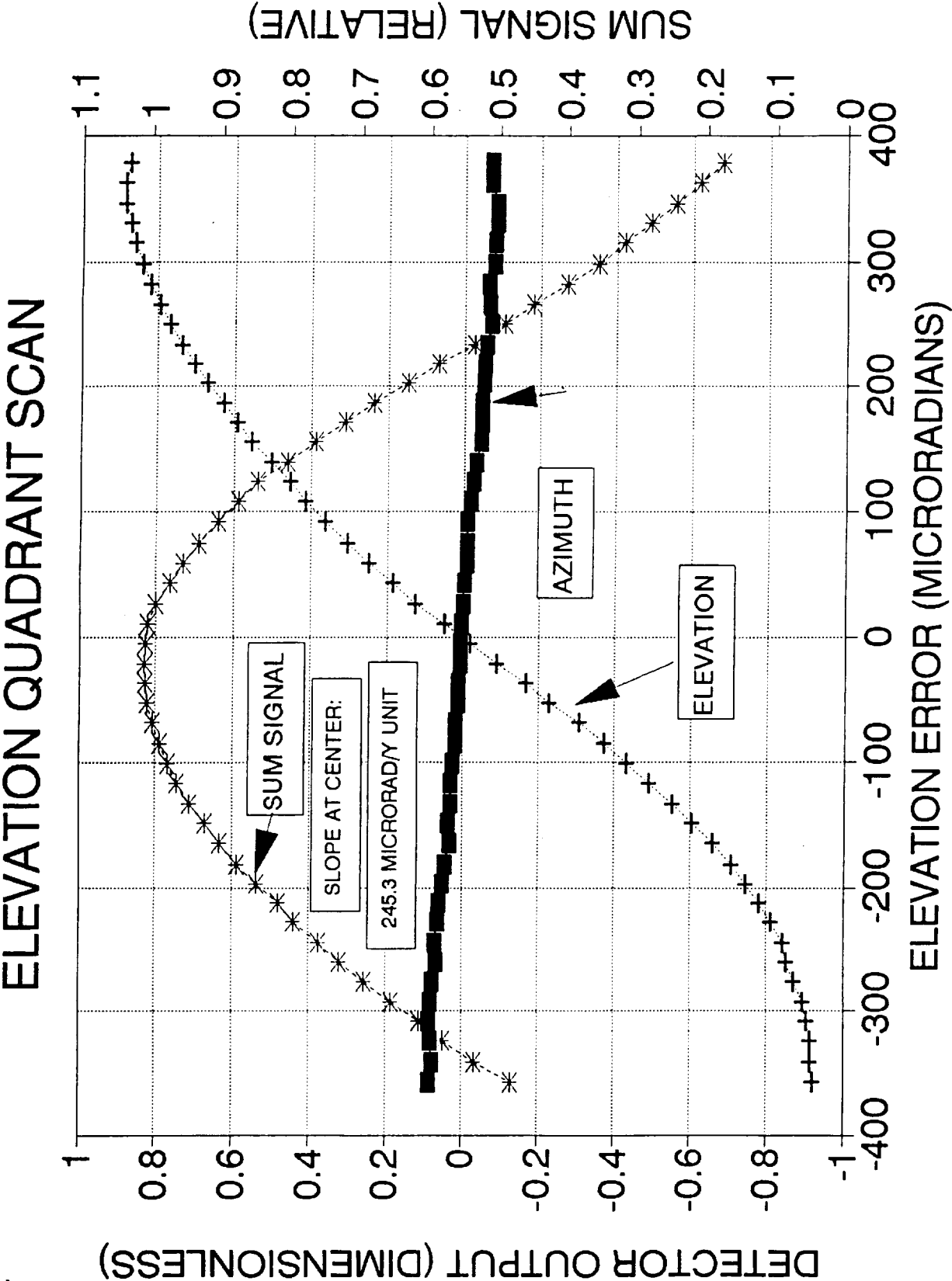


FIGURE 3.1-1

AZIMUTH QUADRANT SCAN

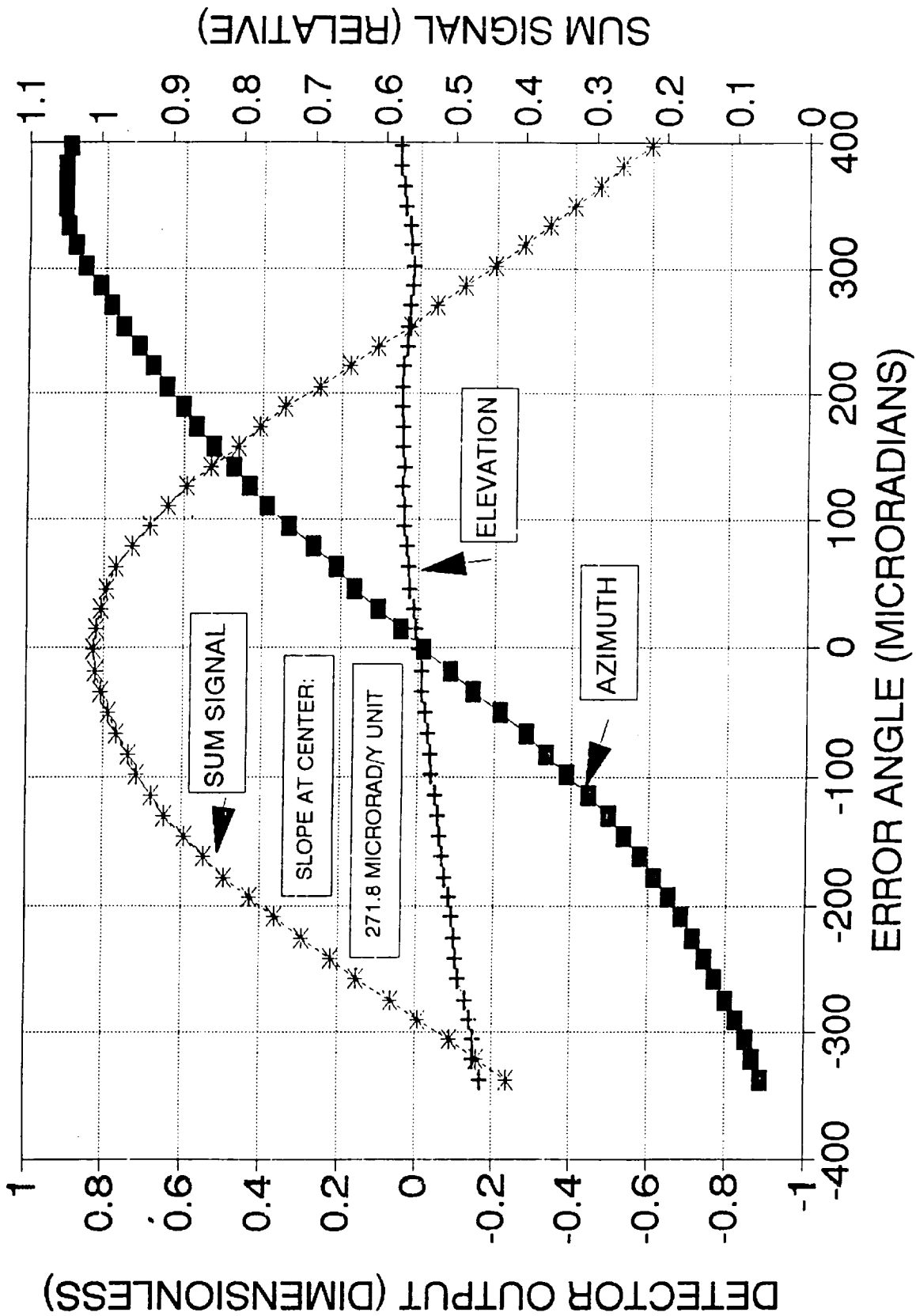


FIGURE 3.1-2

SPOT SHAPE MEASUREMENT

- Channel 1 with Test Aid Diode 2 -

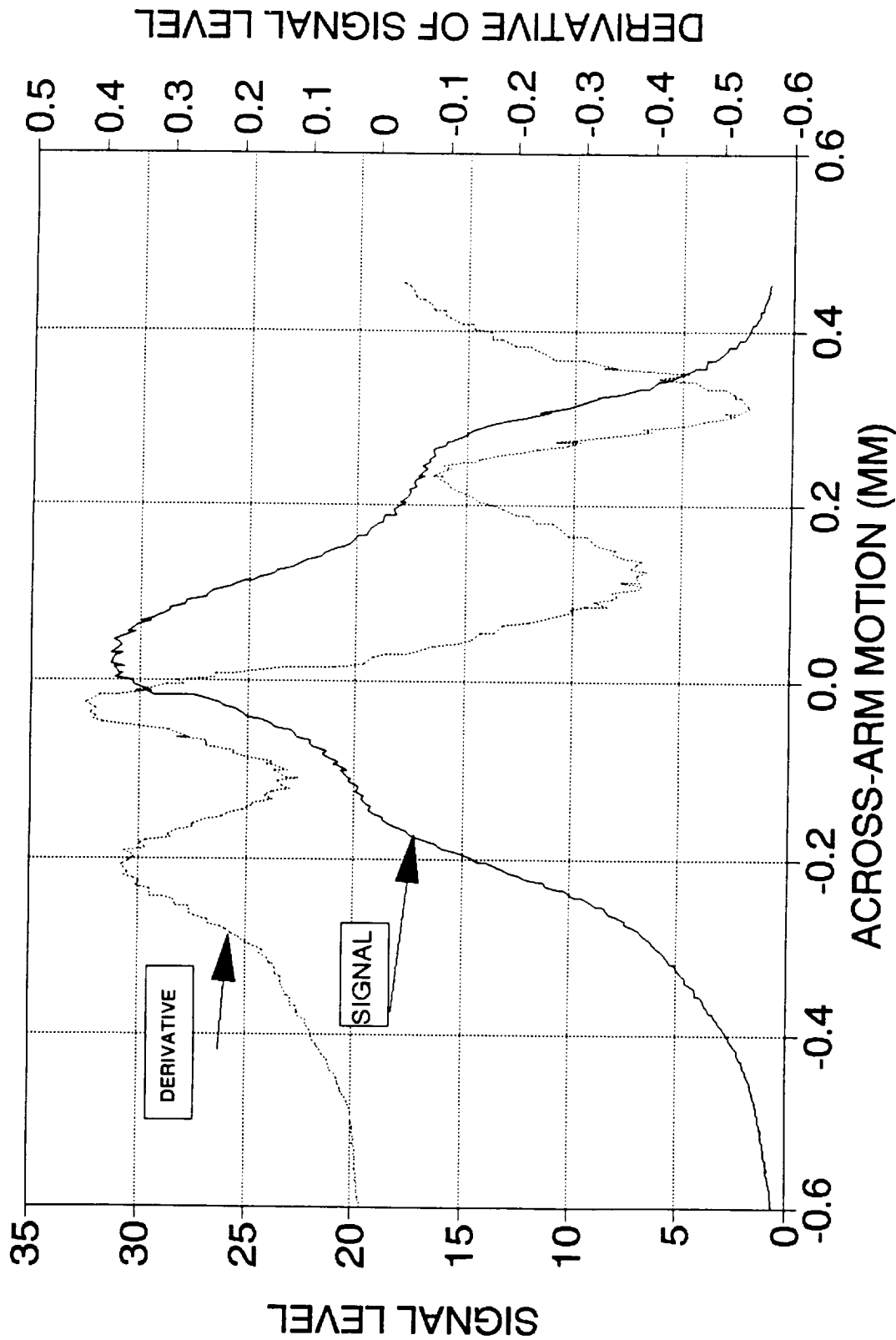


FIGURE 3.1.3

HYSTERESIS PLOT
- Before Adjustment -

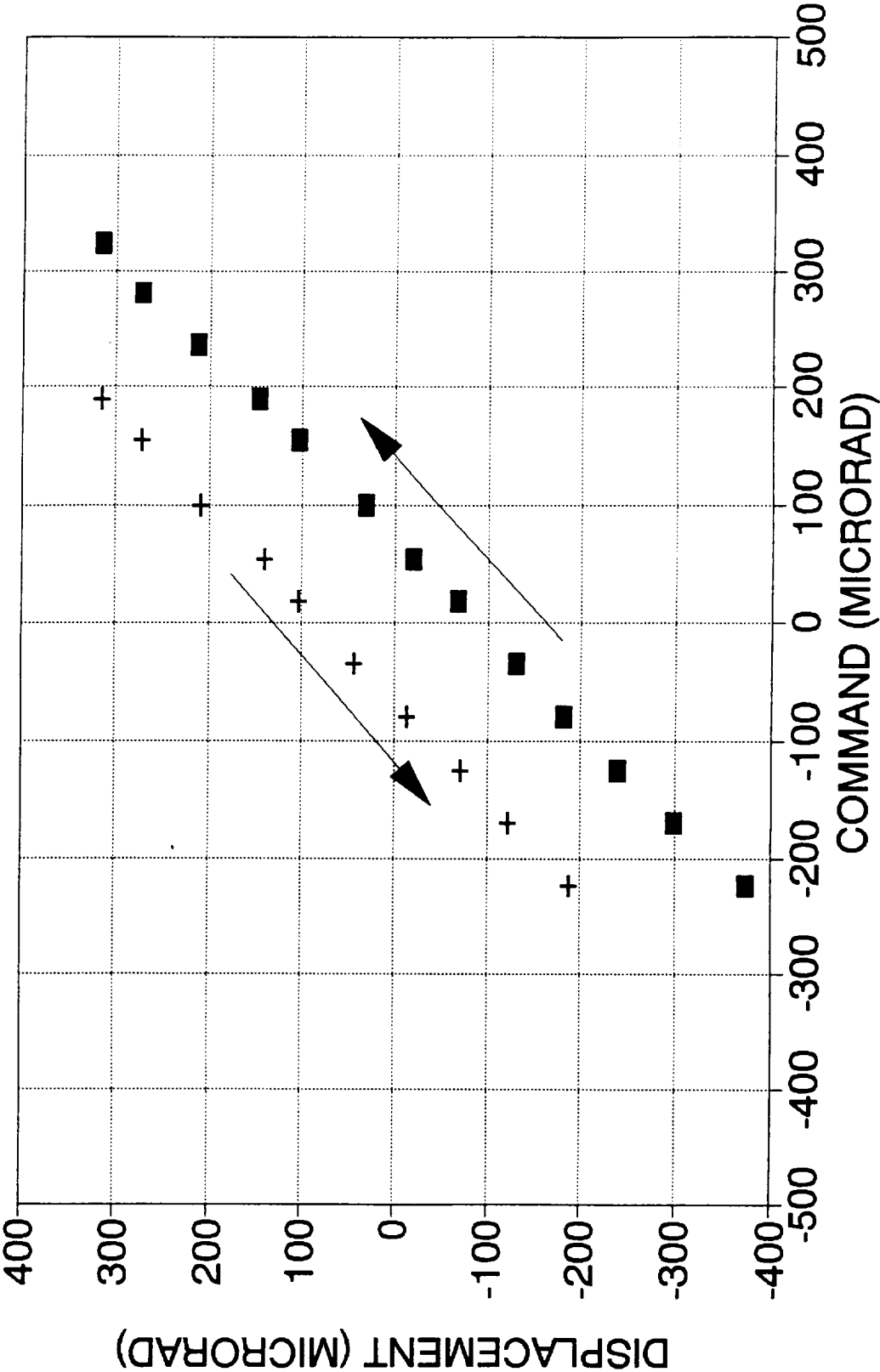


FIGURE 3.2-1

HYSTERESIS PLOT - After Adjustment -

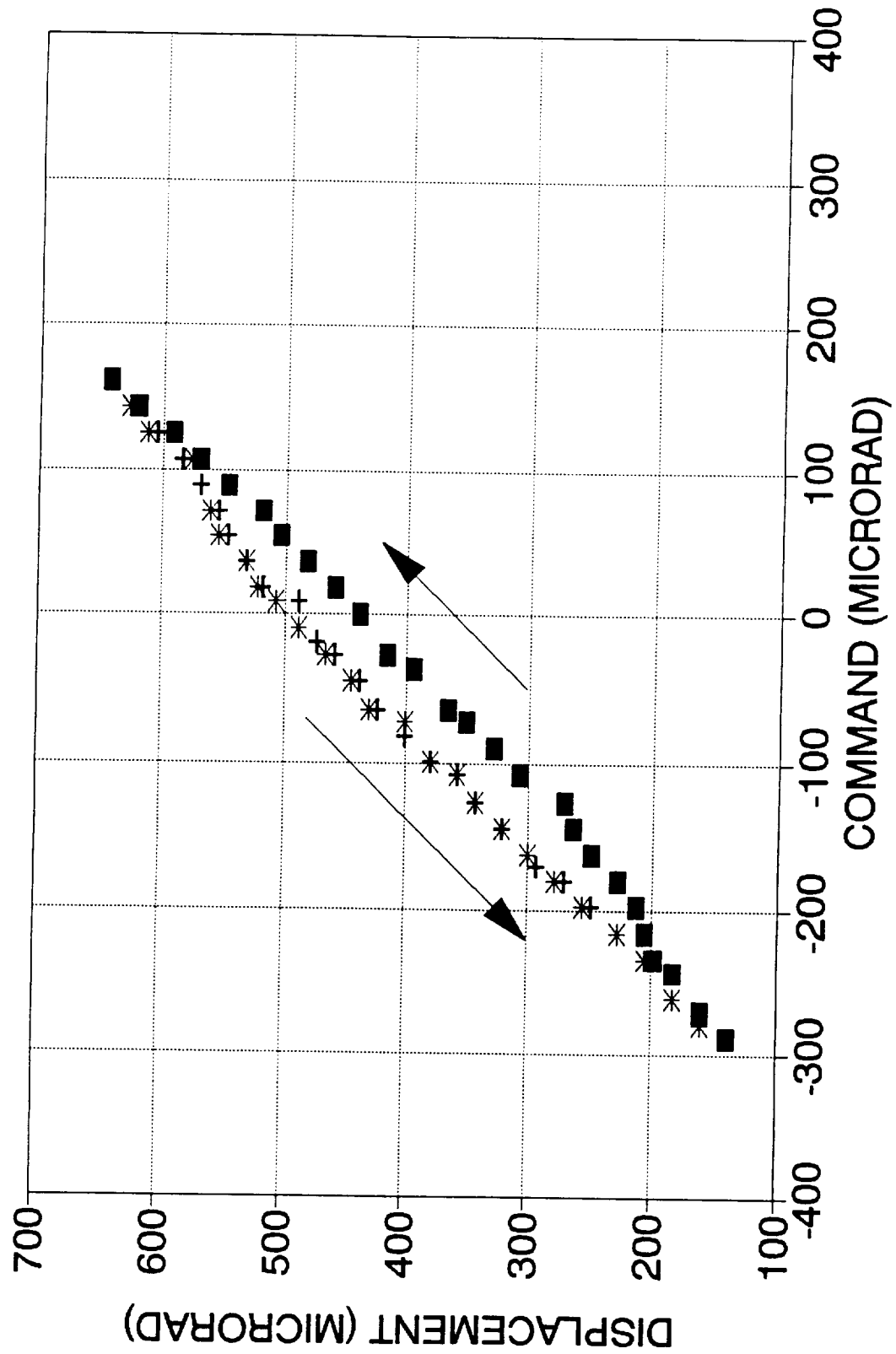


FIGURE 3.2-2

the use of a split, spring-loaded worm which could act as an anti-backlash gear. Additional development of this assembly is recommended in order to improve the automatic tracking performance. The steps required appear to be straightforward extensions of the present work - not great departures from the current design.

Backlash tests were performed on each arm in each axis. Figure 3.2-3 shows a scan which swung the arm across a spot from a test source, then back again. This test was performed with the complete assembly, and, therefore, includes the effects of both the telescope and arm optics. Note that the retrace is displaced about 120 microns from the first trace, indicating a similar amount of backlash in the mechanism. The quadrant data displays some amount of cross-coupling, but the motion is predominantly in *u* - the cross-arm axis of the detector. The amount of backlash observed is more than is tolerable and could have been significantly reduced on disassembly and readjustment for optimum performance. Figure 3.2-4 shows another spot shape scan derived from the same data as for the backlash test. The spot shape is different because of the difference in channel optics and in location in the field.

3.2.2 Slewing rate - The motor control chip accepts maximum values for velocity and acceleration which result in what are referred to as "trapezoidal " moves. The name derives from a plot of speed versus time for a point-to-point move. The speed increases linearly at the maximum acceleration until the maximum velocity is reached, then stays constant until close to the terminus of the move, when the speed decreases linearly at the maximum deceleration, and the motor position arrives at the terminus as the speed reaches zero. The computations for this move are accomplished by the motor control chip, which also issues the commands to accomplish the move. Figure 3.2-5 shows a step response of the motors which compares the response of the large and small motors under various conditions after initial assembly. The small motors were having trouble reacting fast enough when required to work against the other disk or move the arm. We disassembled the set-up and carefully cleaned out any residual machining debris caught in the lubricant with the initial run in and re-ran the test. The result in Figure 3.2-6 revealed that (1) the large motor was not hindered by the additional friction due to its higher torque capability, and (2) the small motor can provide adequate response under all circumstances when properly prepared.

BACKLASH TEST
- Channel 2.5 with Test Aid Diode 2 -

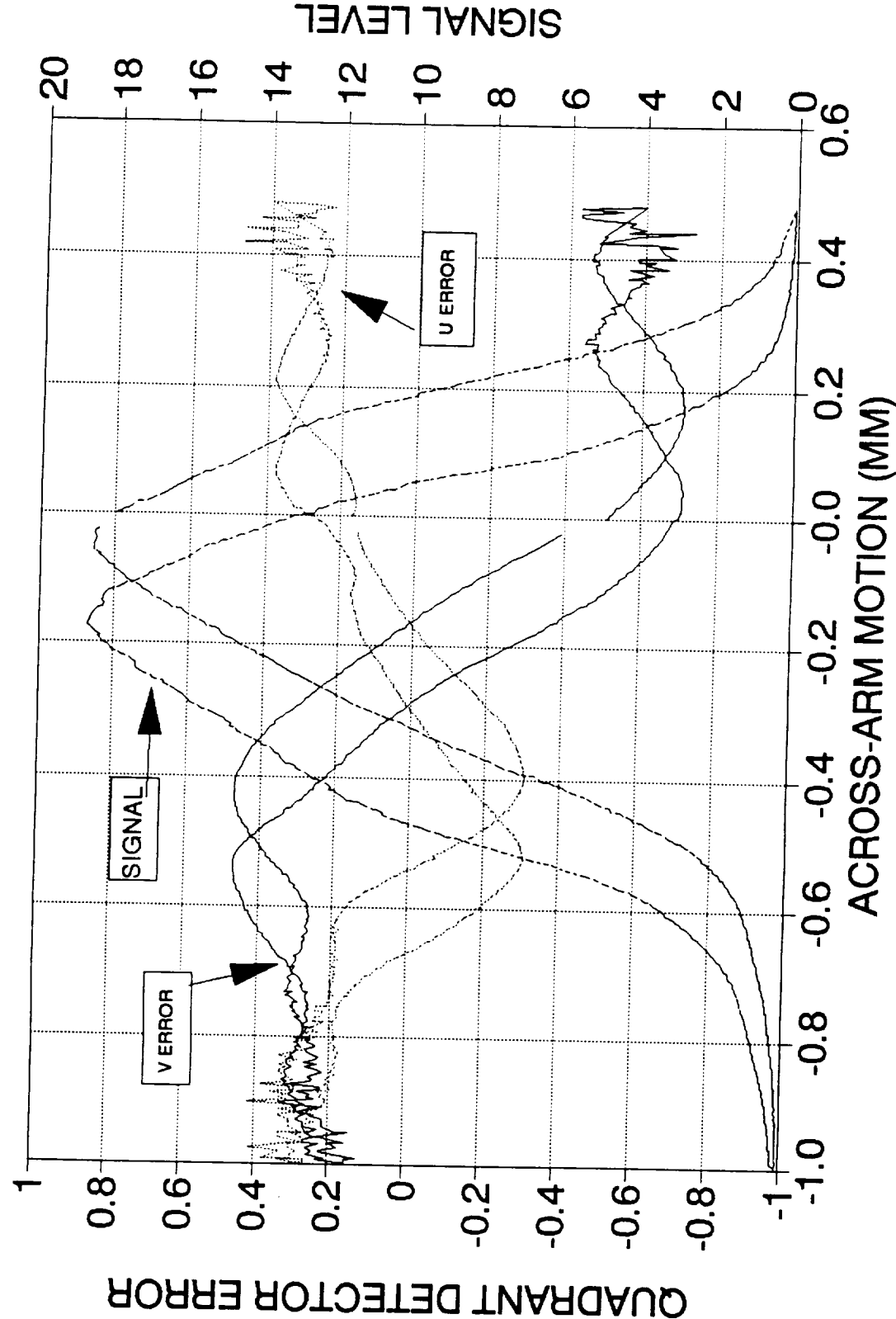


FIGURE 3.2-3

SPOT SHAPE MEASUREMENT

- Channel 2 with Test Aid Diode 2 -

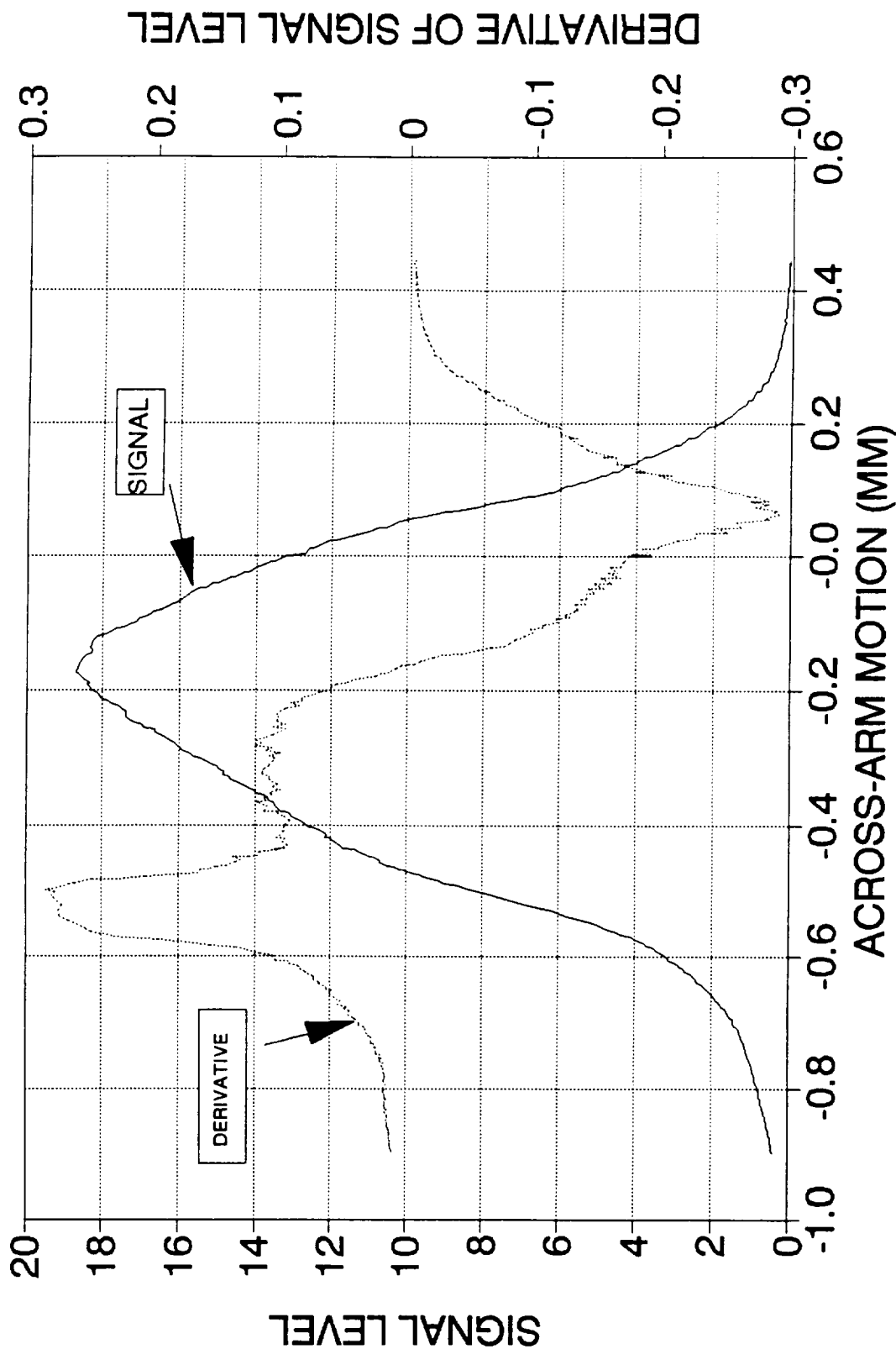


FIGURE 3.2-4

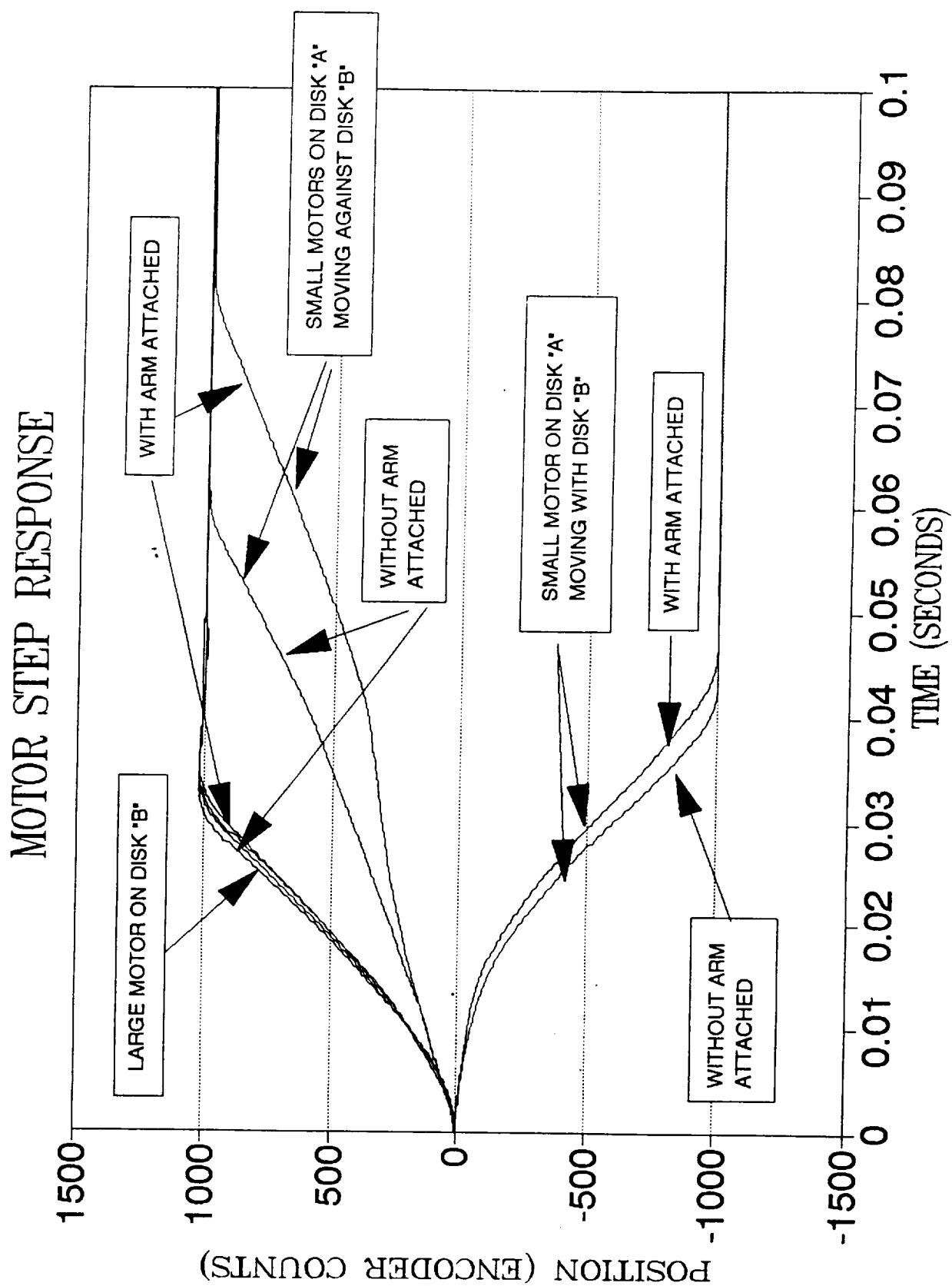


FIGURE 3.2-5

MOTOR STEP RESPONSE - After Cleaning -

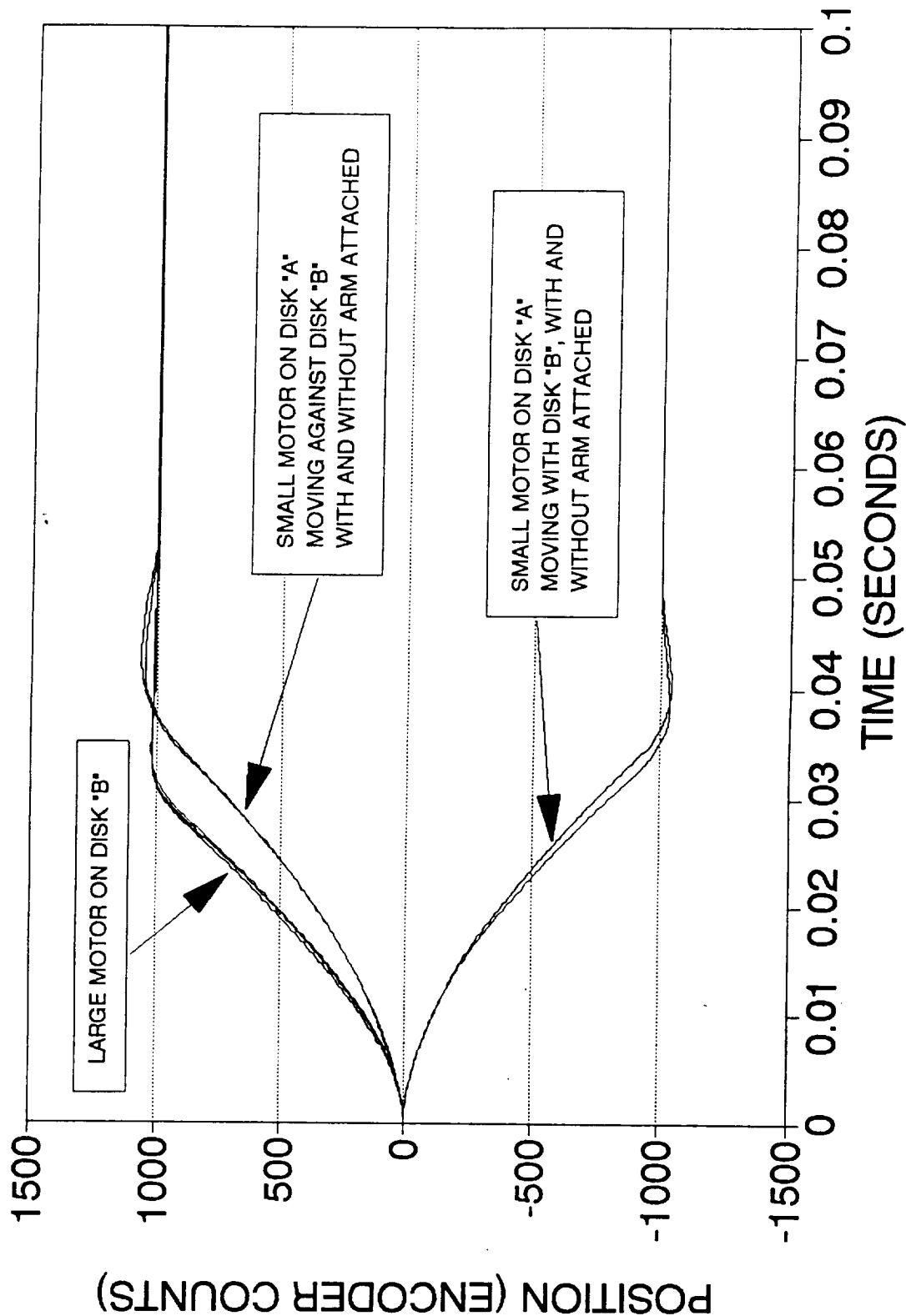


FIGURE 3.2-6

The slewing rate in the far field is a complex function of the trajectory across the field. The most significant occurrence of maximum velocity is when the satellite being tracked passes through the center of the field of view. In the limiting case an outage occurs until the disks can swing through 180° and pick up the signal on the other side of center. We designed for 10° per second and tested 11.4° per second and 7.7° per second, depending on what hexadecimal number was loaded into the chip for maximum velocity. For the prototype we utilized the slower rate to avoid a potential problem overrunning the limit switches on initial calibration. In the space system, the maximum slew rate can be set up to at least 30° per second, resulting in a maximum outage time of 6 seconds for those extreme cases passing directly through the center of the field of view. When another channel is available, hand-off could occur to avoid any appreciable outage. Additional testing would reveal the maximum reasonable value for the slew rate so that we could determine under what circumstances the complexity of the hand-off solution would be superior to the brute-force technique of operating at maximum slew rate.

3.3 Acquisition Time - The primary parameters which make up acquisition time - slew rate, scan pattern, the delay inserted to allow for round-trip optical signal propagation - were either verified or are self-evident. The statistical characteristics of the process having to do with signal fluctuations at the threshold of detectability were beyond the scope of this exercise, but have been explored extensively elsewhere. We feel that the estimates of acquisition time provided in the design rational are conservative.

3.4 Tracking Error - Figure 3.4-1 & -2 shows the tracking error for a stationary target taken on both the across-arm and the along-arm components. The backlash caused some rumble in the motion which is illustrated by the cross-arm component. Further experimentation with the mechanical adjustments and worm/worm gear interface should reduce this effect and result in a smoother, quieter operation.

3.5 Weight Analysis - Table 3.5-1 presents a weight analysis of the space design as compared to the implementation of the prototype. We conclude that the 150-pound goal weight is feasible when lightweighting is utilized in the structural design.

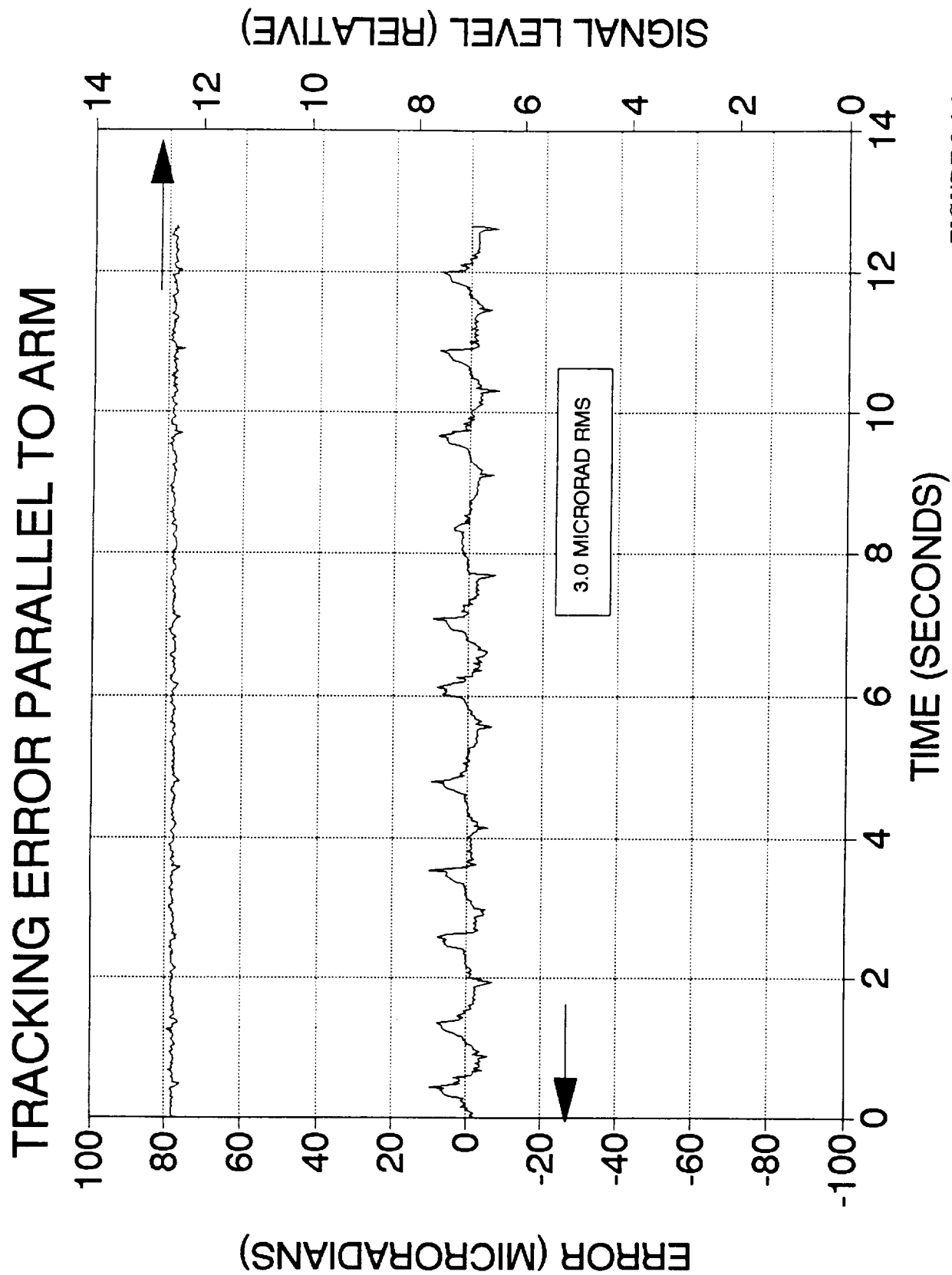


FIGURE 3.4-1

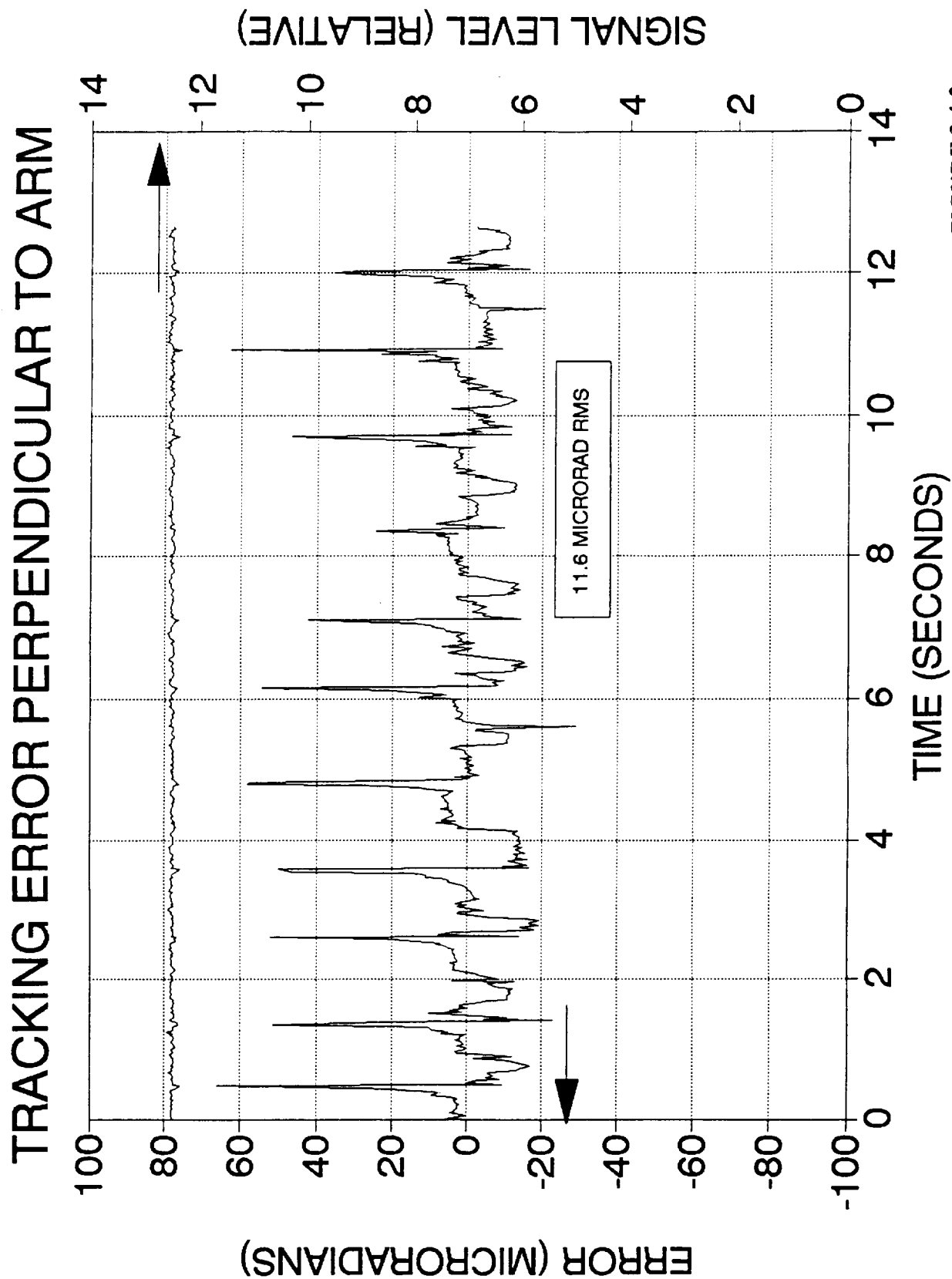


FIGURE 3.4-2

MULTIPLE-ACCESS TERMINAL WEIGHT ANALYSIS

SUBASSEMBLY	WEIGHT (LBS)		COMMENTS
	SPACE DESIGN WITH 6 CHANNELS	PROTOTYPE DESIGN WITH 6 CHANNELS	
DISKS	12	42	Be REPLACES BRASS AND DISK HEIGHT IS OPTIMIZED IN SPACE DESIGN
ARMS	2	2	Be NOT JUSTIFIED
MOTORS/ENCODERS	6	10	ASSUMES PROTOTYPE DESIGN HAS EXCESS TORQUE MARGIN
TRANSMITTER MODULES	3	9	MATERIALS AND CONFIGURATION OPTIMIZED FOR SPACE
STRUCTURE/HARDWARE	23	66	ASSUMES A 70% WEIGHT SAVINGS ON MAIN STRUCTURE DUE TO USE OF LIGHT-WEIGHTING TECHNIQUES AND MATERIALS, PLUS 6 LBS FOR HARDWARE
TELESCOPE	90	115	Be REPLACES Al IN SPACE DESIGN OF STRUCTURE; GLASS REPLACES PLASTIC FOR SPACE OPTICAL ELEMENTS.
ELECTRONICS AND CABLES	12	25	SPACE DESIGN OPTIMIZED
TOTALS	148	269*	

*ACTUAL PROTOTYPE WITH 3 CHANNELS IS ESTIMATED TO WEIGH APPROXIMATELY 200 LBS.

FIGURE 3.5-1

4.0 AREAS FOR FURTHER DEVELOPMENT

The design, development, assembly and testing of the OMA prototype model demonstrated the feasibility of the basic concept and its advantages. A number of issues became clear as to what is needed to enable the OMA to reach its full potential and to be useful space hardware. The following paragraphs discuss these issues.

First, the telescope used in the OMA was necessarily composed of plastic lenses due to cost considerations. This approach adequately demonstrated the concept, but in space, special glass must be used to avoid radiation darkening and to avoid cold-flow shape changes when in space for long periods. We accomplished a glass design to ensure that the same wide field of view can be achieved. In fact, the telescope performance in terms of spot diameters in the focal plane will be better in glass, and this will help tracking performance. Any next step in development must include glass lenses in order to properly evaluate the terminal performance capability.

Second, custom arm optics are required to achieve the vertical spacing required of the six-channel space design. The three-channel prototype design was accomplished with standard optics. Small custom optics allow less depth, less arm width to reduce channel interference possibilities, and overall lighter weight. The next development should include at least one custom optics arm to illustrate the space design capability.

Third, more effort is needed to improve the backlash from the gearing which will smooth out the tracking from some peak errors that accrue. These errors are still within the error budget but cause noise and make the ratio of peak tracking error to rms tracking much greater than it needs to be. The trade-off between less backlash and more friction torque can be efficiently made using the prototype as a tool. Time and money limits prevented further attention in this effort.

Fourth, the tracking bandwidth probably should be raised from 40Hz to about 60Hz to further improve the tracking performance.

Effort should be given to ensuring the optical transmission efficiency and needed performance is obtained on both transmit and receive. On transmit, the proper beamwidth of the transmitter should be established, and any minor optical design

changes or hardware adjustments to achieve required performance should be accomplished. The receiver optics in the arm would benefit from careful measurement for transmission efficiency, image quality, and field of view and adjustment to optimize the output.

The above discussion covers the major areas for further development. The potential and demonstrated advantages of the OMA terminal can be fully realized with a development model based on the achievements of this program and incorporation of the improvements discussed here.

5.0 CONCLUSIONS

The OMA prototype represents a first step towards operational multi-access optical communications terminal development. The concept was shown to be feasible in a low-cost demonstration program which points the way to improvements for operational implementation. The basic design is sound and would benefit from a second program which would provide a thoroughgoing evaluation of the prototype implementation from an optical, mechanical, and electronic viewpoint as a lead-in to a six-channel engineering model. This evaluation program would quantify the performance capability of the present implementation and identify where design improvements are necessary to meet space system requirements and where the present design is adequate. This information would guide the next phase toward activities which are most cost-effective in achieving the space design.

The Phase II program was successful in that a prototype was delivered that demonstrated the key aspects and special advantages of this unique approach to simultaneous multi-access operation. The program showed that the basic advantages of size, weight, power, and Earth-viewing swept volume are realizable. They are substantial when compared to RF implementations or to separate optical transceivers. A patent has been granted for the concept.

The ability of user satellites to have small-size optical transceivers to work with the multi-access terminal is attractive because of the reduced burden on the user satellite. It is recommended that communications applications involving wide bandwidth simultaneously from various satellites to a relay satellite seriously consider this approach as a significantly cost-effective technical solution.

APPENDIX A

TELESCOPE OPTICAL DESIGN SPECIFICATIONS

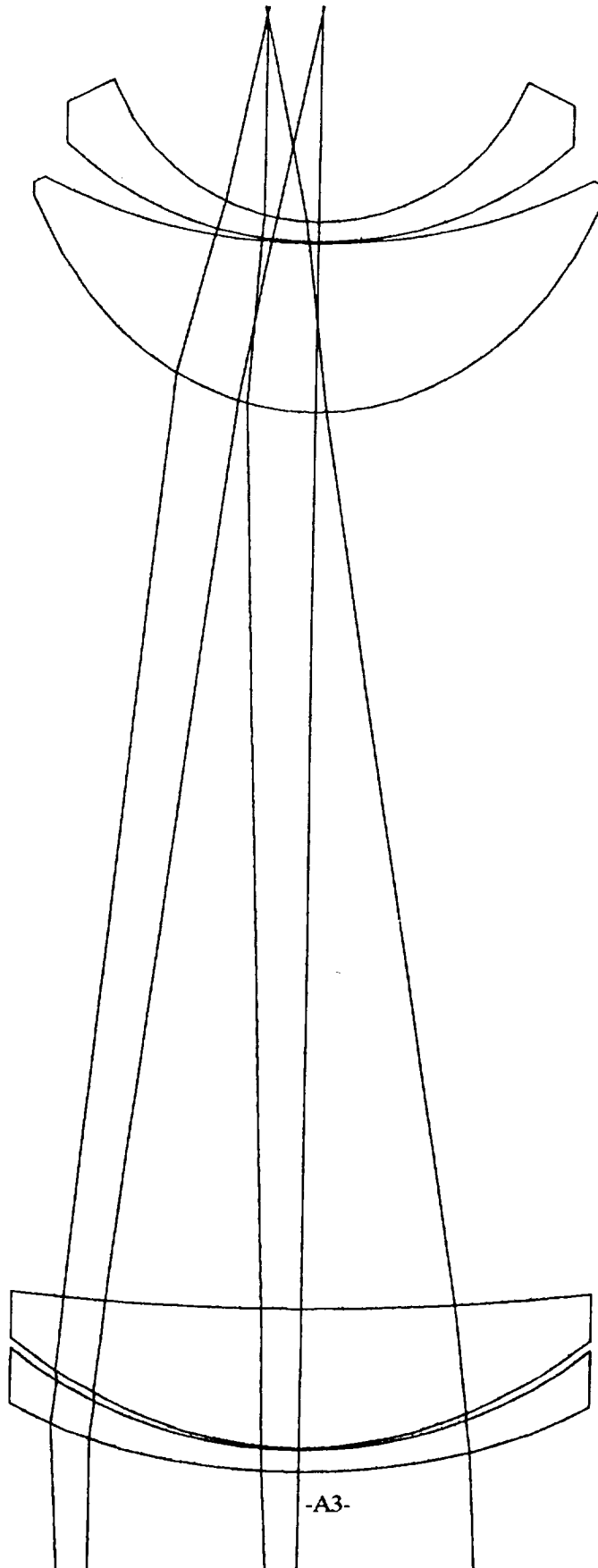
	Page
Telescope Elements with Ray Trace	A3
Ray Trace and Surface Curvature Specifications	A5
Sag Calculations for Individual Elements	A24

THIS PAGE WAS INTENTIONALLY LEFT BLANK

07-05-1991

LOT NASA TELESCOPE VIGNETTED PLASTIC

04:30:45



File ldtplas

Zoom Position 1

SCALE = 0.25

THIS PAGE WAS INTENTIONALLY LEFT BLANK

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE AT 1060 NM WAVELENGTH

File ldtplas 11:08:00

02-14-1991

ZOOM POSITION 1

EFL = 627.483

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57182	1.57597	1.57182	-0.00415	190.58	POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48222	1.48466	1.48222	-0.00245	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48222	1.48466	1.48222	-0.00245	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57182	1.57597	1.57182	-0.00415	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.000000	31899.000	1.00000	1.00000	1.00000	0.00000	6501.32	
12	S	0.000000	-31904.000	1.00000	1.00000	1.00000	0.00000	116.66	
13	S	0.000000	1.949	1.00000	1.00000	1.00000	0.00000	116.66	

ASPHERIC SURFACE 2 CC = 2.477985

A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23

ASPHERIC SURFACE 4 CC = -1.247590

A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23

ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)

A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
	-0.19969		0.04986	-0.01416	-0.03584			
	-0.09998		0.07627	-0.00368	-0.00851			

CHIEF RAY								
-0.00103			32.80324			0.417	0.592	-0.251%
0.19907			0.07775	-0.00609	-0.01198			
0.17949			0.00651	-0.00685	-0.00756			
0.15946			-0.03360	-0.00653	-0.00384			
0.13912			-0.05378	-0.00562	-0.00086			
0.09930			-0.05975	-0.00324	0.00278			
0.05917			-0.04020	-0.00118	0.00367			
-0.06125			0.04603	-0.00120	-0.00976			
-0.10137			0.11702	-0.00433	-0.01992			
-0.14071			0.20302	-0.01070	-0.03304			
-0.16149			0.21961	-0.01516	-0.04132			
-0.18155			0.19138	-0.01936	-0.05025			
-0.20055			0.11954	-0.02238	-0.05961			
-0.19978	-0.00134	-0.00402	0.04571	-0.00899	-0.03587			
-0.17600	-0.00127	0.07360	0.03750	-0.00800	-0.02743			
-0.13995	-0.00119	0.08343	0.02058	-0.00492	-0.01699			
-0.10010	-0.00112	0.05048	0.00561	-0.00223	-0.00853			

ZOOM POSITION 2

EFL = 627.483

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

3 S	0.003544	12.700	1.57182	1.57597	1.57182-0.00415	190.58	POLYST
4 A	0.004324	1.000	1.00000	1.00000	1.00000 0.00000	193.87	
5 S	0.000633	81.524	1.48222	1.48466	1.48222-0.00245	193.08	ACRYLC
6 A	0.006593	543.944	1.00000	1.00000	1.00000 0.00000	170.50	
7 S	0.002784	102.098	1.48222	1.48466	1.48222-0.00245	164.06	ACRYLC
8 S	0.004486	1.000	1.00000	1.00000	1.00000 0.00000	151.30	
9 S	0.007514	12.700	1.57182	1.57597	1.57182-0.00415	123.88	POLYST
10 S	0.000000	130.000	1.00000	1.00000	1.00000 0.00000	117.52	
11 S	0.00000031899	0.000	1.00000	1.00000	1.00000 0.00000	6501.32	
12 S	0.000000%-31904	0.000	1.00000	1.00000	1.00000 0.00000	116.66	
13 S	0.000000	1.949	1.00000	1.00000	1.00000 0.00000	116.66	

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
-0.19969			0.04986	-0.01416	-0.03584			
-0.13991			0.11692	-0.00754	-0.01698			

CHIEF RAY -0.00205 65.25814 -0.430 0.173 -1.052%

0.19826	0.12553	-0.00176	0.01159
0.18082	0.04902	-0.00323	0.01322
0.16049	-0.00682	-0.00361	0.01443
0.13813	-0.03681	-0.00307	0.01496
0.09953	-0.03873	-0.00146	0.01392
0.05891	-0.01573	-0.00035	0.01030
-0.06303	0.05263	-0.00106	-0.01655
-0.10367	0.14769	-0.00504	-0.03150
-0.14219	0.20683	-0.01216	-0.04893
-0.16474	0.19081	-0.01671	-0.06073
-0.18511	0.16483	-0.02030	-0.07251
-0.20232	0.18265	-0.02320	-0.08339
-0.19982	-0.00249	-0.14106	0.07209
-0.17672	-0.00239	-0.04290	0.06486
-0.13995	-0.00227	-0.00309	0.04179
-0.10048	-0.00217	-0.01555	0.01664

ZOOM POSITION 3
 EFL = 627.483

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1 S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61		
2 A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34		
3 S	0.003544	12.700	1.57182	1.57597	1.57182-0.00415	190.58	POLYST		
4 A	0.004324	1.000	1.00000	1.00000	1.00000 0.00000	193.87			
5 S	0.000633	81.524	1.48222	1.48466	1.48222-0.00245	193.08	ACRYLC		
6 A	0.006593	543.944	1.00000	1.00000	1.00000 0.00000	170.50			
7 S	0.002784	102.098	1.48222	1.48466	1.48222-0.00245	164.06	ACRYLC		
8 S	0.004486	1.000	1.00000	1.00000	1.00000 0.00000	151.30			
9 S	0.007514	12.700	1.57182	1.57597	1.57182-0.00415	123.88	POLYST		
10 S	0.000000	130.000	1.00000	1.00000	1.00000 0.00000	117.52			
11 S	0.00000031899	0.000	1.00000	1.00000	1.00000 0.00000	6501.32			
12 S	0.000000%-31904	0.000	1.00000	1.00000	1.00000 0.00000	116.66			

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

13 S 0.000000 1.949 1.00000 1.00000 1.00000 0.00000 116.66

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000
 L M X Y OPD COLOR S T DIST(%)
 -0.19969 0.04986 -0.01416 -0.03584
 -0.15984 0.12738 -0.01000 -0.02241

CHIEF RAY -0.00304 96.92833 -0.915 -0.155 -2.472%
 0.19779 0.03167 -0.00231 0.03478
 0.17875 -0.02492 -0.00231 0.03394
 0.15781 -0.04897 -0.00148 0.03241
 0.13743 -0.04607 -0.00048 0.03030
 0.09723 -0.00947 0.00068 0.02446
 0.05966 0.01291 0.00053 0.01699
 -0.06588 0.03325 -0.00060 -0.02355
 -0.10371 0.05731 -0.00246 -0.04142
 -0.14185 0.02371 -0.00413 -0.06264
 -0.16487 0.03433 -0.00464 -0.07727
 -0.18587 0.12763 -0.00625 -0.09200
 -0.20156 0.08258 -0.00832 -0.10363
 -0.20020 -0.00284 -0.22885 0.03963 0.01175 -0.03573
 -0.17814 -0.00287 -0.11858 0.04820 0.00803 -0.02793
 -0.14014 -0.00292 -0.04832 0.04450 0.00523 -0.01695
 -0.10124 -0.00298 -0.04757 0.02520 0.00344 -0.00869

ZOOM POSITION 4
 EFL = 627.483

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	GLASS
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57182	1.57597	1.57182	-0.00415	190.58	
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	POLYST
5	S	0.000633	81.524	1.48222	1.48466	1.48222	-0.00245	193.08	
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	ACRYLC
7	S	0.002784	102.098	1.48222	1.48466	1.48222	-0.00245	164.06	
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	POLYST
9	S	0.007514	12.700	1.57182	1.57597	1.57182	-0.00415	123.88	
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	POLYST
11	S	0.000000	31899.000	1.00000	1.00000	1.00000	0.00000	6501.32	
12	S	0.000000	-31904.000	1.00000	1.00000	1.00000	0.00000	116.66	

13 S 0.000000 1.949 1.00000 1.00000 1.00000 0.00000 118.25

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

Defocus =		U' =							
L	M	X	Y	OPD	COLOR	S	T	DIST(%)	
	-0.19969		0.04986	-0.01416	-0.03584				
	-0.17977		0.11205	-0.01245	-0.02869				
CHIEF RAY	-0.00367		117.07032			0.299	-0.077	-4.019%	
	0.17954		0.04506	0.00699	0.04752				
	0.15564		0.04058	0.00601	0.04344				
	0.14243		0.04538	0.00545	0.04092				
	0.12483		0.05329	0.00458	0.03725				
	0.09603		0.05833	0.00294	0.03056				
	0.05610		0.03814	0.00092	0.01980				
	-0.06281		0.03820	-0.00079	-0.02515				
	-0.10180		0.05982	-0.00283	-0.04553				
	-0.13616		0.07907	-0.00501	-0.06658				
	-0.14681		0.11211	-0.00601	-0.07383				
	-0.15956		0.16762	-0.00780	-0.08299				
	-0.18130		-0.03959	-0.01074	-0.09906				
-0.19916	-0.00069	-0.00202	-0.03419	-0.01153	-0.03449				
-0.18002	-0.00122	0.07080	-0.01468	-0.01080	-0.02788				
-0.13936	-0.00217	0.11434	0.01284	-0.00658	-0.01638				
-0.10226	-0.00285	0.07775	0.01406	-0.00293	-0.00868				

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE AT 860NM WAVELENGTH File ldtplas 10:49:05 02-14-1991
 ZOOM POSITION 1
 EFL = 625.402

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	GLASS
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57597	1.57720	1.57597-0.00123	190.58		POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48466	1.48527	1.48466-0.00061	193.08		ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48466	1.48527	1.48466-0.00061	164.06		ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57597	1.57720	1.57597-0.00123	123.88		POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.000000	31899.000	1.00000	1.00000	1.00000	0.00000	6503.48	
12	S	0.000000	-31904.000	1.00000	1.00000	1.00000	0.00000	116.66	
13	S	0.000000	0.291	1.00000	1.00000	1.00000	0.00000	116.66	

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
-0.19974			0.00782	-0.01189	-0.00674			
-0.09999			0.07005	-0.00352	-0.00153			

CHIEF RAY -0.00102 32.69479 0.418 0.593 -0.250%

0.19913			0.09423	-0.00566	-0.00205			
0.17953			0.01709	-0.00668	-0.00122			
0.15949			-0.02777	-0.00653	-0.00053			
0.13914			-0.05146	-0.00570	0.00000			
0.09931			-0.06087	-0.00333	0.00062			
0.05917			-0.04151	-0.00121	0.00073			
-0.06125			0.04188	-0.00112	-0.00182			
-0.10137			0.10328	-0.00391	-0.00373			
-0.14073			0.17362	-0.00946	-0.00624			
-0.16151			0.17955	-0.01320	-0.00785			
-0.18157			0.13922	-0.01648	-0.00962			
-0.20059			0.05339	-0.01838	-0.01149			
-0.19983	-0.00134	-0.04542	0.03665	-0.00676	-0.00675			
-0.17603	-0.00127	0.04374	0.03022	-0.00661	-0.00511			
-0.13997	-0.00118	0.06736	0.01578	-0.00434	-0.00311			
-0.10010	-0.00111	0.04438	0.00310	-0.00208	-0.00154			

ZOOM POSITION 2
 EFL = 625.402

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	GLASS
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57597	1.57720	1.57597-0.00123	190.58		POLYST

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

4 A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5 S	0.000633	81.524	1.48466	1.48527	1.48466	-0.00061	193.08	ACRYLC
6 A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7 S	0.002784	102.098	1.48466	1.48527	1.48466	-0.00061	164.06	ACRYLC
8 S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9 S	0.007514	12.700	1.57597	1.57720	1.57597	-0.00123	123.88	POLYST
10 S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11 S	0.00000031899.000	1.00000	1.00000	1.00000	1.00000	0.000006503.48		
12 S	0.000000%-31904.000	1.00000	1.00000	1.00000	1.00000	0.00000	116.66	
13 S	0.000000	0.291	1.00000	1.00000	1.00000	0.00000	116.66	

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
	-0.19974		0.00782	-0.01189	-0.00674			
	-0.13993		0.10059	-0.00696	-0.00310			
CHIEF RAY	-0.00204		65.04438			-0.424	0.173	-1.048%
	0.19831		0.11282	-0.00340	0.00248			
	0.18087		0.03529	-0.00465	0.00275			
	0.16052		-0.02073	-0.00474	0.00294			
	0.13815		-0.04984	-0.00390	0.00299			
	0.09953		-0.04824	-0.00185	0.00271			
	0.05891		-0.02027	-0.00045	0.00197			
	-0.06302		0.04526	-0.00091	-0.00312			
	-0.10366		0.12615	-0.00434	-0.00597			
	-0.14219		0.16511	-0.01026	-0.00938			
	-0.16474		0.13306	-0.01369	-0.01172			
	-0.18512		0.08751	-0.01593	-0.01410			
	-0.20234		0.08373	-0.01732	-0.01633			
-0.19987	-0.00248	-0.18066	0.05366	0.00643	-0.00678			
-0.17675	-0.00238	-0.07161	0.04995	0.00366	-0.00517			
-0.13996	-0.00226	-0.01835	0.03193	0.00233	-0.00313			
-0.10049	-0.00216	-0.02132	0.01136	0.00161	-0.00156			

ZOOM POSITION 3
 EFL = 625.402

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57597	1.57720	1.57597	-0.00123	190.58	POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48466	1.48527	1.48466	-0.00061	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48466	1.48527	1.48466	-0.00061	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57597	1.57720	1.57597	-0.00123	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.00000031899.000	1.00000	1.00000	1.00000	1.00000	0.000006503.48		
12	S	0.000000%-31904.000	1.00000	1.00000	1.00000	1.00000	0.00000	116.66	

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

13 S 0.000000 0.291 1.00000 1.00000 1.00000 0.00000 116.66

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
-0.19974			0.00782	-0.01189	-0.00674			
-0.15987			0.10385	-0.00902	-0.00413			

CHIEF RAY -0.00303 96.61543 -0.912 -0.135 -2.463%

0.19787	-0.01894	-0.00682	0.00675
0.17881	-0.07098	-0.00590	0.00655
0.15785	-0.08929	-0.00416	0.00621
0.13745	-0.08022	-0.00240	0.00577
0.09723	-0.03064	-0.00012	0.00462
0.05966	0.00304	0.00030	0.00319
-0.06584	0.02439	-0.00043	-0.00445
-0.10366	0.03367	-0.00169	-0.00791
-0.14178	-0.02558	-0.00202	-0.01211
-0.16481	-0.04109	-0.00112	-0.01507
-0.18581	0.02694	-0.00085	-0.01807
-0.20146	-0.00352	-0.00138	-0.02048
-0.20024	-0.00283	-0.26677	0.01110
-0.17817	-0.00286	-0.14624	0.02503
-0.14015	-0.00291	-0.06298	0.02921
-0.10124	-0.00297	-0.05346	0.01674

ZOOM POSITION 4
 EFL = 625.402

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57597	1.57720	1.57597	-0.00123	190.58	POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48466	1.48527	1.48466	-0.00061	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48466	1.48527	1.48466	-0.00061	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57597	1.57720	1.57597	-0.00123	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.000000	31899.000	1.00000	1.00000	1.00000	0.00000	6503.48	
12	S	0.000000	-31904.000	1.00000	1.00000	1.00000	0.00000	116.66	

13 S 0.000000 0.291 1.00000 1.00000 1.00000 0.00000 117.83

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
	-0.19974		0.00782	-0.01189	-0.00674			
	-0.17981		0.07994	-0.01092	-0.00534			
CHIEF RAY	-0.00366		<u>116.70510</u>			0.281	0.013	-3.999
	0.17959		-0.04771	-0.00019	0.00886			
	0.15563		-0.03813	0.00089	0.00807			
	0.14241		-0.02503	0.00131	0.00759			
	0.12479		-0.00574	0.00159	0.00690			
	0.09598		0.01784	0.00137	0.00565			
	0.05606		0.02024	0.00050	0.00366			
	-0.06275		0.03073	-0.00069	-0.00473			
	-0.10174		0.03255	-0.00210	-0.00867			
	-0.13614		0.01518	-0.00280	-0.01286			
	-0.14682		0.03183	-0.00303	-0.01432			
	-0.15960		0.07045	-0.00368	-0.01618			
	-0.18124		-0.07965	-0.00476	-0.01952			
-0.19919	-0.00072	-0.04115	-0.07250	-0.00938	-0.00664			
-0.18003	-0.00124	0.04133	-0.04612	-0.00928	-0.00532			
-0.13935	-0.00218	0.09865	-0.00714	-0.00593	-0.00307			
-0.10225	-0.00285	0.07019	0.00257	-0.00269	-0.00161			

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE AT 820NM WAVELENGTH File ldtplas 10:38:13 02-14-1991
 ZOOM POSITION 1
 EFL = 625.007

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57720	1.57597	1.57720	0.00123	190.58	POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48527	1.48466	1.48527	0.00061	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48527	1.48466	1.48527	0.00061	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57720	1.57597	1.57720	0.00123	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.00000031899	0.000	1.00000	1.00000	1.00000	0.00000	6504.12	
12	S	0.000000%-31904	0.000	1.00000	1.00000	1.00000	0.00000	116.66	
13	S	0.000000	-0.004	1.00000	1.00000	1.00000	0.00000	116.66	

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
-0.19976			-0.00731	-0.01109	0.00673			
-0.09999			0.06787	-0.00347	0.00153			

CHIEF RAY -0.00102 32.67420 0.418 0.590 -0.250%

0.19914			0.10167	-0.00539	0.00204			
0.17954			0.02215	-0.00653	0.00121			
0.15950			-0.02463	-0.00646	0.00053			
0.13914			-0.04977	-0.00568	-0.00000			
0.09931			-0.06075	-0.00334	-0.00062			
0.05917			-0.04173	-0.00122	-0.00073			
-0.06125			0.04038	-0.00109	0.00182			
-0.10137			0.09857	-0.00377	0.00373			
-0.14073			0.16351	-0.00903	0.00623			
-0.16152			0.16562	-0.01252	0.00784			
-0.18159			0.12086	-0.01548	0.00960			
-0.20061			0.03005	-0.01699	0.01147			
-0.19984	-0.00134	-0.06054	0.03380	-0.00595	0.00674			
-0.17605	-0.00127	0.03297	0.02795	-0.00611	0.00510			
-0.13998	-0.00118	0.06162	0.01430	-0.00413	0.00311			
-0.10010	-0.00111	0.04215	0.00232	-0.00202	0.00154			

ZOOM POSITION 2
 EFL = 625.007

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57720	1.57597	1.57720	0.00123	190.58	POLYST

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

4 A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87			
5 S	0.000633	81.524	1.48527	1.48466	1.48527	0.00061	193.08	ACRYLC		
6 A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50			
7 S	0.002784	102.098	1.48527	1.48466	1.48527	0.00061	164.06	ACRYLC		
8 S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30			
9 S	0.007514	12.700	1.57720	1.57597	1.57720	0.00123	123.88	POLYST		
10 S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52			
11 S	0.00000031899	0.000	1.00000	1.00000	1.00000	0.000006504	116.66			
12 S	0.000000%-31904	0.000	1.00000	1.00000	1.00000	0.00000	116.66			
13 S	0.000000	-0.004	1.00000	1.00000	1.00000	0.00000	116.66			

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus =	-0.750000	U' =	-0.200000							
L	M	X	Y	OPD	COLOR	S	T	DIST(%)		
-0.19976			-0.00731	-0.01109	0.00673					
-0.13994			0.09487	-0.00675	0.00310					
CHIEF RAY	-0.00204		65.00388			-0.426	0.162	-1.047%		
	0.19832		0.11268	-0.00361	-0.00248					
	0.18087		0.03426	-0.00485	-0.00275					
	0.16052		-0.02241	-0.00491	-0.00294					
	0.13815		-0.05179	-0.00403	-0.00298					
	0.09953		-0.04984	-0.00191	-0.00271					
	0.05891		-0.02093	-0.00046	-0.00197					
	-0.06301		0.04239	-0.00085	0.00311					
	-0.10366		0.11834	-0.00407	0.00597					
	-0.14219		0.15001	-0.00955	0.00936					
	-0.16475		0.11225	-0.01259	0.01171					
	-0.18513		0.06032	-0.01433	0.01408					
	-0.20236		0.05013	-0.01520	0.01630					
-0.19989	-0.00248	-0.19576	0.04791	0.00725	0.00677					
-0.17676	-0.00238	-0.08250	0.04536	0.00419	0.00516					
-0.13997	-0.00226	-0.02417	0.02895	0.00255	0.00312					
-0.10049	-0.00216	-0.02368	0.00978	0.00167	0.00156					

ZOOM POSITION 3
 EFL = 625.007

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57720	1.57597	1.57720	0.00123	190.58	POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48527	1.48466	1.48527	0.00061	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48527	1.48466	1.48527	0.00061	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57720	1.57597	1.57720	0.00123	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.00000031899	0.000	1.00000	1.00000	1.00000	0.000006504	116.66	
12	S	0.000000%-31904	0.000	1.00000	1.00000	1.00000	0.00000	116.66	

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

13 S 0.000000 -0.004 1.00000 1.00000 1.00000 0.00000 116.66

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000
 L M X Y OPD COLOR S T DIST(%)
 -0.19976 -0.00731 -0.01109 0.00673
 -0.15988 0.09557 -0.00868 0.00413

CHIEF RAY -0.00303 96.55641 -0.918 -0.156 -2.461%
 0.19788 -0.02726 -0.00753 -0.00674
 0.17881 -0.07872 -0.00645 -0.00654
 0.15785 -0.09612 -0.00456 -0.00620
 0.13745 -0.08594 -0.00267 -0.00577
 0.09723 -0.03386 -0.00021 -0.00461
 0.05965 0.00195 0.00029 -0.00319
 -0.06583 0.01997 -0.00033 0.00444
 -0.10365 0.02323 -0.00131 0.00789
 -0.14177 -0.04536 -0.00108 0.01209
 -0.16480 -0.06879 0.00037 0.01504
 -0.18581 -0.00842 0.00130 0.01804
 -0.20146 -0.03975 0.00134 0.02044
 -0.20026 -0.00283 -0.28219 0.00230 0.01467 0.00681
 -0.17818 -0.00286 -0.15761 0.01795 0.00993 0.00527
 -0.14015 -0.00291 -0.06913 0.02466 0.00602 0.00315
 -0.10124 -0.00297 -0.05619 0.01428 0.00368 0.00159

ZOOM POSITION 4
 EFL = 625.007

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	GLASS
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57720	1.57597	1.57720	0.00123	190.58	
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48527	1.48466	1.48527	0.00061	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48527	1.48466	1.48527	0.00061	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57720	1.57597	1.57720	0.00123	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.000000	31899.000	1.00000	1.00000	1.00000	0.00000	6504.12	
12	S	0.000000	-31904.000	1.00000	1.00000	1.00000	0.00000	116.66	

13 S 0.000000 -0.004 1.00000 1.00000 1.00000 0.00000 117.83

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

Defocus = -0.750000 U' = -0.200000

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

L	M	X	Y	OPD	COLOR	S	T	DIST(%)
	-0.19976		-0.00731	-0.01109	0.00673			
	-0.17982		0.06854	-0.01038	0.00533			
CHIEF RAY	-0.00366		116.63673			0.268	-0.006	-3.9954
	0.17958		-0.06361	-0.00129	-0.00884			
	0.15562		-0.05132	0.00014	-0.00805			
	0.14239		-0.03663	0.00072	-0.00757			
	0.12477		-0.01517	0.00118	-0.00688			
	0.09596		0.01189	0.00119	-0.00563			
	0.05604		0.01824	0.00047	-0.00365			
	-0.06274		0.02587	-0.00059	0.00472			
	-0.10173		0.01991	-0.00167	0.00865			
	-0.13613		-0.00883	-0.00175	0.01283			
	-0.14682		0.00328	-0.00170	0.01428			
	-0.15960		0.03666	-0.00195	0.01614			
	-0.18124		-0.11087	-0.00228	0.01947			
-0.19920	-0.00073	-0.05721	-0.08369	-0.00844	0.00662			
-0.18004	-0.00125	0.02893	-0.05533	-0.00861	0.00531			
-0.13936	-0.00218	0.09193	-0.01287	-0.00564	0.00306			
-0.10225	-0.00285	0.06682	-0.00063	-0.00257	0.00160			

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE AT 820NM WAVELENGTH File ldtplas 10:07:21 02-14-1991
 ZOOM POSITION 1
 EFL = 625.007

#	TYPE	CURVE	SEPN	INDEX1	INDEX2	INDEX3	DISPN	CLRAD	GLASS
1	S	0.000000	0.000	1.00000	1.00000	1.00000	0.00000	125.61	GLASS
2	A	0.002135	400.000	1.00000	1.00000	1.00000	0.00000	195.34	
3	S	0.003544	12.700	1.57720	1.57597	1.57720	0.00123	190.58	POLYST
4	A	0.004324	1.000	1.00000	1.00000	1.00000	0.00000	193.87	
5	S	0.000633	81.524	1.48527	1.48466	1.48527	0.00061	193.08	ACRYLC
6	A	0.006593	543.944	1.00000	1.00000	1.00000	0.00000	170.50	
7	S	0.002784	102.098	1.48527	1.48466	1.48527	0.00061	164.06	ACRYLC
8	S	0.004486	1.000	1.00000	1.00000	1.00000	0.00000	151.30	
9	S	0.007514	12.700	1.57720	1.57597	1.57720	0.00123	123.88	POLYST
10	S	0.000000	130.000	1.00000	1.00000	1.00000	0.00000	117.52	
11	S	0.000000	31899.000	1.00000	1.00000	1.00000	0.00000	6573.72	
12	S	0.000000	%-31904.000	1.00000	1.00000	1.00000	0.00000	116.66	
13	S	0.000000	-0.003	1.00000	1.00000	1.00000	0.00000	116.66	

ASPHERIC SURFACE 2 CC = 2.477985
 A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

LDT NASA TELESCOPE AT 820NM WAVELENGTH File ldtplas 10:08:10 02-14-1991
 ZOOM POSITION 1
 EFL = 625.007

RADIUS	SEPN	CLR DIAM	MATERIAL
PLANE		251.23	
468.277	400.000	390.68	Air
282.162	12.700	381.15	POLYST
231.275	1.000	387.75	Air
1579.829	81.524	386.16	ACRYLC
151.670	543.944	341.01	Air
359.155	102.098	328.13	ACRYLC
222.913	1.000	302.60	Air
133.093	12.700	247.76	POLYST
PLANE	130.000	235.03	Air
PLANE	31899.000	13147.45	Air
PLANE	%-31904.000	233.32	Air
PLANE	-0.003	233.32	Air

ASPHERIC SURFACE 2 CC = 2.477985

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

A4 = 4.479591E-09 A6 = -8.743832E-14 A8 = 1.936009E-18 A10 = -3.557055E-23
 ASPHERIC SURFACE 4 CC = -1.247590
 A4 = 1.901898E-11 A6 = 7.800049E-14 A8 = -1.635492E-18 A10 = 2.510175E-23
 ASPHERIC SURFACE 6 CC = -0.354793 (ELLIPSE)
 A4 = -3.357945E-09 A6 = 1.277421E-14 A8 = -2.20762E-18 A10 = 3.734123E-23

LDT NASA TELESCOPE AT 820NM WAVELENGTH File ldtplas 10:08:43 02-14-1991
 ZOOM POSITION 1
 EFL = 625.007

#	Type	Xrel	Yrel	X	Y	L	M
1	P	0.000	1.000	0.00000	125.00238	0.000000	0.000000
2	MD	0.000	1.000	0.00000	125.00238	0.000000	0.000000
3	M	0.000	0.500	0.00000	62.50119	0.000000	0.000000
4	CD	0.000	3.000	0.00000	1.26883	0.000000	0.052336
5	MD	0.000	-0.998	0.00000	-123.46299	0.000000	0.052336
6	M	0.000	-0.900	0.00000	-111.23332	0.000000	0.052336
7	M	0.000	-0.800	0.00000	-98.73308	0.000000	0.052336
8	M	0.000	-0.698	0.00000	-86.04344	0.000000	0.052336
9	M	0.000	-0.500	0.00000	-61.23237	0.000000	0.052336
10	M	0.000	-0.300	0.00000	-36.23189	0.000000	0.052336
11	M	0.000	0.300	0.00000	38.76954	0.000000	0.052336
12	M	0.000	0.500	0.00000	63.77002	0.000000	0.052336
13	M	0.000	0.696	0.00000	88.31071	0.000000	0.052336
14	M	0.000	0.800	0.00000	101.27073	0.000000	0.052336
15	M	0.000	0.900	0.00000	113.77097	0.000000	0.052336
16	MD	0.000	0.995	0.00000	125.61438	0.000000	0.052336
17	S	0.999	0.000	124.89674	1.26883	0.000000	0.052336
18	S	0.880	0.000	110.00210	1.26883	0.000000	0.052336
19	S	0.699	0.000	87.42772	1.26883	0.000000	0.052336
20	S	0.500	0.000	62.50119	1.26883	0.000000	0.052336

ZOOM POSITION 2
 EFL = 625.007

#	Type	Xrel	Yrel	X	Y	L	M
21	PZ	0.000	1.000	0.00000	125.00238	0.000000	0.000000
22	MD	0.000	1.000	0.00000	125.00238	0.000000	0.000000
23	M	0.000	0.700	0.00000	87.50167	0.000000	0.000000
24	CD	0.000	6.000	0.00000	2.42560	0.000000	0.104529
25	MD	0.000	-0.986	0.00000	-120.79125	0.000000	0.104529
26	M	0.000	-0.900	0.00000	-110.07654	0.000000	0.104529
27	M	0.000	-0.800	0.00000	-97.57631	0.000000	0.104529
28	M	0.000	-0.690	0.00000	-83.82619	0.000000	0.104529
29	M	0.000	-0.500	0.00000	-60.07559	0.000000	0.104529
30	M	0.000	-0.300	0.00000	-35.07512	0.000000	0.104529
31	M	0.000	0.300	0.00000	39.92631	0.000000	0.104529
32	M	0.000	0.500	0.00000	64.92679	0.000000	0.104529
33	M	0.000	0.689	0.00000	88.59774	0.000000	0.104529
34	M	0.000	0.800	0.00000	102.42751	0.000000	0.104529
35	M	0.000	0.900	0.00000	114.92774	0.000000	0.104529
36	MD	0.000	0.985	0.00000	125.52866	0.000000	0.104529
37	S	0.995	0.000	124.40148	2.42560	0.000000	0.104529
38	S	0.880	0.000	110.00210	2.42560	0.000000	0.104529
39	S	0.697	0.000	87.08103	2.42560	0.000000	0.104529
40	S	0.500	0.000	62.50119	2.42560	0.000000	0.104529

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

ZOOM POSITION 3

EFL = 625.007

#	Type	Xrel	Yrel	X	Y	L	M
41	PZ	0.000	1.000	0.00000	125.00238	0.000000	0.000000
42	MD	0.000	1.000	0.00000	125.00238	0.000000	0.000000
43	M	0.000	0.800	0.00000	100.00191	0.000000	0.000000
44	CD	0.000	9.000	0.00000	5.60687	0.000000	0.156435
45	MD	0.000	-0.961	0.00000	-114.48861	0.000000	0.156435
46	M	0.000	-0.870	0.00000	-103.14520	0.000000	0.156435
47	M	0.000	-0.770	0.00000	-90.64496	0.000000	0.156435
48	M	0.000	-0.673	0.00000	-78.45996	0.000000	0.156435
49	M	0.000	-0.480	0.00000	-54.39427	0.000000	0.156435
50	M	0.000	-0.300	0.00000	-31.89384	0.000000	0.156435
51	M	0.000	0.300	0.00000	43.10759	0.000000	0.156435
52	M	0.000	0.480	0.00000	65.60801	0.000000	0.156435
53	M	0.000	0.661	0.00000	88.20391	0.000000	0.156435
54	M	0.000	0.770	0.00000	101.85870	0.000000	0.156435
55	M	0.000	0.870	0.00000	114.35894	0.000000	0.156435
56	MD	0.000	0.944	0.00000	123.60264	0.000000	0.156435
57	S	0.989	0.000	123.61288	5.60687	0.000000	0.156435
58	S	0.880	0.000	110.00210	5.60687	0.000000	0.156435
59	S	0.692	0.000	86.52901	5.60687	0.000000	0.156435
60	S	0.500	0.000	62.50119	5.60687	0.000000	0.156435

ZOOM POSITION 4

EFL = 625.007

#	Type	Xrel	Yrel	X	Y	L	M
61	PZ	0.000	1.000	0.00000	125.00238	0.000000	0.000000
62	MD	0.000	1.000	0.00000	125.00238	0.000000	0.000000
63	M	0.000	0.900	0.00000	112.50214	0.000000	0.000000
64	CD	0.000	11.000	0.00000	4.40677	0.000000	0.190809
65	MD	0.000	-0.829	0.00000	-99.19891	0.000000	0.190809
66	M	0.000	-0.720	0.00000	-85.59495	0.000000	0.190809
67	M	0.000	-0.660	0.00000	-78.09481	0.000000	0.190809
68	M	0.000	-0.580	0.00000	-68.11722	0.000000	0.190809
69	M	0.000	-0.450	0.00000	-51.84431	0.000000	0.190809
70	M	0.000	-0.270	0.00000	-29.34388	0.000000	0.190809
71	M	0.000	0.270	0.00000	38.15741	0.000000	0.190809
72	M	0.000	0.450	0.00000	60.65784	0.000000	0.190809
73	M	0.000	0.610	0.00000	80.65822	0.000000	0.190809
74	M	0.000	0.660	0.00000	86.90834	0.000000	0.190809
75	M	0.000	0.720	0.00000	94.40848	0.000000	0.190809
76	MD	0.000	0.820	0.00000	106.90872	0.000000	0.190809
77	S	0.973	0.000	121.67773	4.40677	0.000000	0.190809
78	S	0.880	0.000	110.00210	4.40677	0.000000	0.190809
79	S	0.681	0.000	85.17441	4.40677	0.000000	0.190809
80	S	0.500	0.000	62.50119	4.40677	0.000000	0.190809

LDT NASA TELESCOPE AT 820NM WAVELENGTH File ldtplas 10:09:03 02-14-1991

ZOOM POSITION 1

EFL = 625.007

OPTIMIZATION CONDITIONS

ANGLE SOLVE OFF

EFL	BACK FOCUS	Ybar	Mbar'	Track	Glass	Length	Th-ness
-----	------------	------	-------	-------	-------	--------	---------

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

WEIGHTING FACTORS

1.60	25.00	0.00	500.00	0.00	1.00	0.00	0.00	TARGE
625.000	0.000	0.0000	0.0000	0.000	209.000	0.000	0.000	

RAY NO	LABEL	WT	WT	WT
1	P	4.0		
2	MD	10.0	100.0	
3	M	10.0	100.0	
4	CD	4.0	4.0	150.0
5	MD	10.0	100.0	
6	M	10.0	100.0	
7	M	10.0	100.0	
8	M	10.0	100.0	
9	M	10.0	100.0	
10	M	10.0	100.0	
11	M	10.0	100.0	
12	M	10.0	100.0	
13	M	10.0	100.0	
14	M	10.0	100.0	
15	M	10.0	100.0	
16	MD	10.0	100.0	
17	S	10.0	10.0	100.0
18	S	10.0	10.0	100.0
19	S	10.0	10.0	100.0
20	S	10.0	10.0	100.0

#	VARIABLES	CURVE	EDGE	CLR RAD LIMIT
1		0.00000	0.000	428.302
2	CEA4A6A8A10	0.00214	2.000	200.000
3	C	0.00354	12.700	200.000
4	CEA4A6A8A10	0.00432	1.000	200.000
5	C D	0.00063	12.700	428.302
6	CEA4A6A8A10 D	0.00659	2.000	428.302
7	C D	0.00278	13.000	200.000
8	C	0.00449	1.000	200.000
9	C	0.00751	12.700	428.302
10		0.00000	0.000	200.000
11		0.00000	70.000	18000.000
12		0.00000%	-40000.000	12000.000
13		0.00000%	-40000.000	428.302

ZOOM POSITION 2

EFL = 625.007

OPTIMIZATION CONDITIONS

ANGLE SOLVE OFF

EFL	BACK FOCUS	Ybar	Mbar'	Track	Glass	Length	Th-ness
WEIGHTING FACTORS							
1.60	25.00	0.00	500.00	0.00	1.00	0.00	0.00
625.000	0.000	0.0000	0.0000	0.000	209.000	0.000	0.000

RAY NO	LABEL	WT	WT	WT
21	PZ	4.0		
22	MD	10.0	100.0	
23	M	10.0	100.0	
24	CD	4.0	4.0	150.0

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

25	MD	10.0	100.0	
26	M	10.0	100.0	
27	M	10.0	100.0	
28	M	10.0	100.0	
29	M	10.0	100.0	
30	M	10.0	100.0	
31	M	10.0	100.0	
32	M	10.0	100.0	
33	M	10.0	100.0	
34	M	10.0	100.0	
35	M	10.0	100.0	
36	MD	10.0	100.0	
37	S	10.0	10.0	100.0
38	S	10.0	10.0	100.0
39	S	10.0	10.0	100.0
40	S	10.0	10.0	100.0

#	VARIABLES	CURVE	EDGE	CLR	RAD	LIMIT
1		0.00000	0.000		428.302	
2		0.00214	2.000		200.000	
3		0.00354	12.700		200.000	
4		0.00432	1.000		200.000	
5		0.00063	12.700		428.302	
6		0.00659	2.000		428.302	
7		0.00278	13.000		200.000	
8		0.00449	1.000		200.000	
9		0.00751	12.700		428.302	
10		0.00000	0.000		200.000	
11		0.00000	70.000		18000.000	
12		0.00000%	-40000.000		12000.000	
13		0.00000%	-40000.000		428.302	

ZOOM POSITION 3

EFL = 625.007

OPTIMIZATION CONDITIONS

ANGLE SOLVE OFF

EFL	BACK FOCUS	Ybar	Mbar'	Track	Glass	Length	Th-ness
1.60	25.00	0.00	500.00	0.00	1.00	0.00	0.00
625.000	0.000	0.0000	0.0000	0.000	209.000	0.000	0.000

TARGETS

RAY NO	LABEL	WT	WT	WT
41	PZ	4.0		
42	MD	10.0	100.0	
43	M	10.0	100.0	
44	CD	4.0	4.0	150.0
45	MD	10.0	100.0	
46	M	10.0	100.0	
47	M	10.0	100.0	
48	M	10.0	100.0	
49	M	10.0	100.0	
50	M	10.0	100.0	
51	M	10.0	100.0	
52	M	10.0	100.0	
53	M	10.0	100.0	
54	M	10.0	100.0	
55	M	10.0	100.0	

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

56	MD	10.0	100.0	
57	S	10.0	10.0	100.0
58	S	10.0	10.0	100.0
59	S	10.0	10.0	100.0
60	S	10.0	10.0	100.0

#	VARIABLES	CURVE	EDGE	CLR RAD LIMIT
1		0.00000	0.000	428.302
2		0.00214	2.000	200.000
3		0.00354	12.700	200.000
4		0.00432	1.000	200.000
5		0.00063	12.700	428.302
6		0.00659	2.000	428.302
7		0.00278	13.000	200.000
8		0.00449	1.000	200.000
9		0.00751	12.700	428.302
10		0.00000	0.000	200.000
11		0.00000	70.000	18000.000
12		0.00000%	-40000.000	12000.000
13		0.00000%	-40000.000	428.302

ZOOM POSITION 4

EFL = 625.007

OPTIMIZATION CONDITIONS

ANGLE SOLVE OFF

EFL	BACK FOCUS	Ybar	Mbar'	Track	Glass	Length	Th-ness
1.60	25.00	0.00	500.00	0.00	1.00	0.00	0.00
625.000	0.000	0.0000	0.0000	0.000	209.000	0.000	0.000

RAY NO	LABEL	WT	WT	WT
61	PZ	4.0		
62	MD	10.0	100.0	
63	M	10.0	100.0	
64	CD	4.0	4.0	150.0
65	MD	10.0	100.0	
66	M	10.0	100.0	
67	M	10.0	100.0	
68	M	10.0	100.0	
69	M	10.0	100.0	
70	M	10.0	100.0	
71	M	10.0	100.0	
72	M	10.0	100.0	
73	M	10.0	100.0	
74	M	10.0	100.0	
75	M	10.0	100.0	
76	MD	10.0	100.0	
77	S	10.0	10.0	100.0
78	S	10.0	10.0	100.0
79	S	10.0	10.0	100.0
80	S	10.0	10.0	100.0

#	VARIABLES	CURVE	EDGE	CLR RAD LIMIT
1		0.00000	0.000	428.302
2		0.00214	2.000	200.000
3		0.00354	12.700	200.000
4		0.00432	1.000	200.000

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

5	0.00063	12.700	428.302
6	0.00659	2.000	428.302
7	0.00278	13.000	200.000
8	0.00449	1.000	200.000
9	0.00751	12.700	428.302
10	0.00000	0.000	200.000
11	0.00000	70.000	18000.000
12	0.00000%	-40000.000	12000.000
13	0.00000%	-40000.000	428.302

12/04/90

General Lens Specifications

Total Integrated Scatter < 1 % per surface

Surface Roughness < .005 microns rms

Wavelengths of operation .82 microns and .86 microns

Wedge angle < 6 arc minutes

All aspheric surfaces:

Slope angle within .0005 radians of nominal value defined by
surface equation for all points within clear aperture.

Slope angle to be measured over .5 inch span.

Figure deviation $\pm .0075$ inches max from nominal curve.

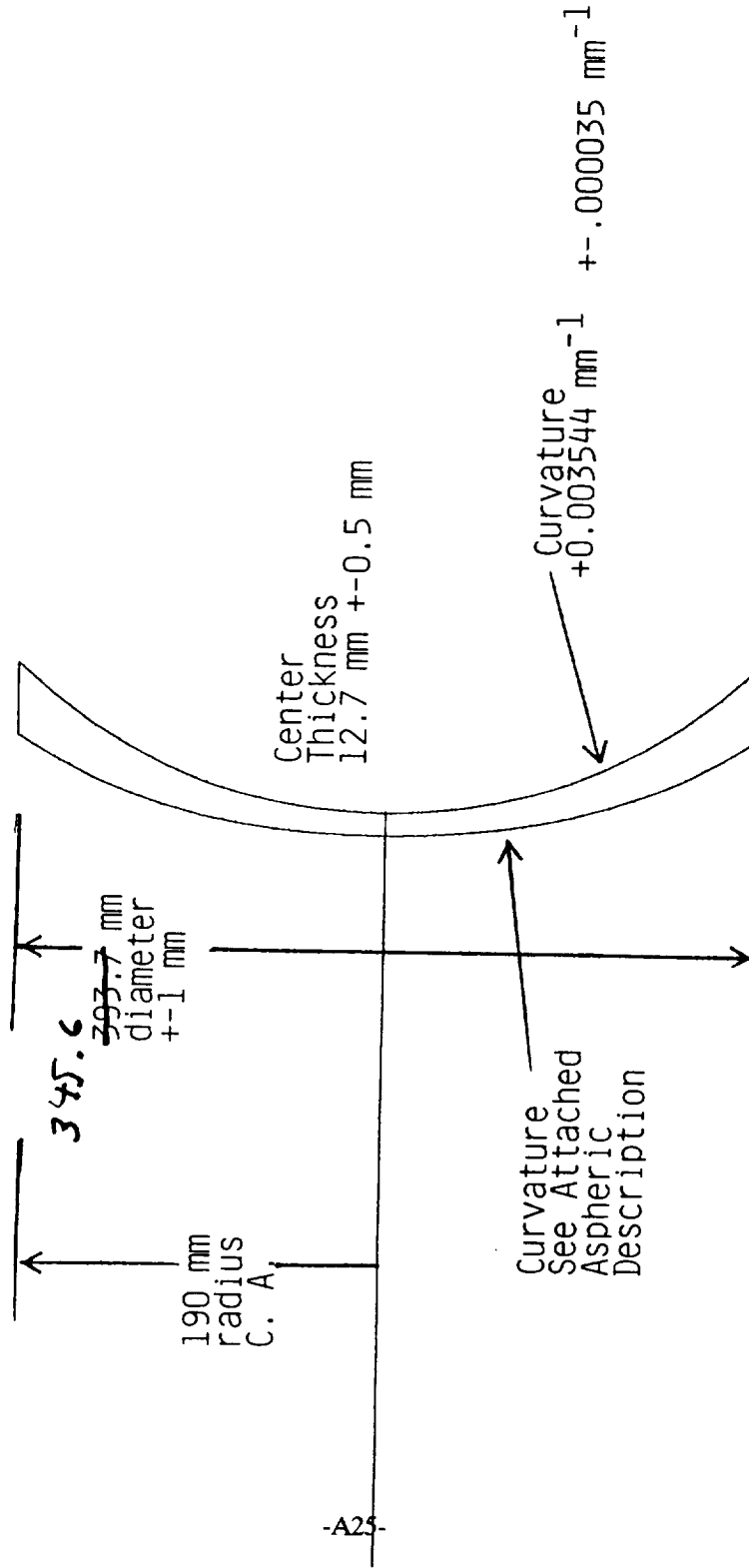
PERSPECTIVE DISPLAYS, INC.

Sheet 1 of 1

12-04-1990

LDT NASA TELESCOPE ELEMENT 001

MDP 7/23/91



Material: Optical Grade Polystyrene

PERSPECTIVE DISPLAYS, INC.

Marvin L. Rand

Sheet 1 of 5

File ldtplas

SCALE = 0.25

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 001 SURFACE 1
Sag. vs. radial distance from optical axis

M.L.Pund 12/04/90
Perspective Displays,
Inc.

$$Z = (C*(R**2))/(1+\text{sqrt}(1-(1+K)*(C**2)*(R**2))) + A4*(R**4) + A6*(R**6) + A8*(R**8) + A10*(R**10)$$

C =	2.1350000E-03 /mm	Radius of Curvature
K =	2.4779850E+00	Conic constant
A4 =	4.4795910E-09	Fourth order aspheric coefficient
A6 =	-8.7438320E-14	Sixth order aspheric coefficient
A8 =	1.9360090E-18	Eighth order aspheric coefficient
A10 =	-3.5570550E-23	Tenth order aspheric coefficient

R = radial distance from optical axis (mm)
Z = Sag. (mm)

R (mm)	Z (mm)	R (inches)	Z (inches)
0.0000000	0.0000000	0.0000000	0.0000000
1.2700000	0.0017218	0.0500000	0.0000678
2.5400000	0.0068874	0.1000000	0.0002712
3.8100000	0.0154978	0.1500000	0.0006101
5.0800000	0.0275541	0.2000000	0.0010848
6.3500000	0.0430584	0.2500000	0.0016952
7.6200000	0.0620131	0.3000000	0.0024415
8.8900000	0.0844211	0.3500000	0.0033237
10.1600000	0.1102861	0.4000000	0.0043420
11.4300000	0.1396120	0.4500000	0.0054965
12.7000000	0.1724034	0.5000000	0.0067875
13.9700000	0.2086656	0.5500000	0.0082152
15.2400000	0.2484042	0.6000000	0.0097797
16.5100000	0.2916254	0.6500000	0.0114813
17.7800000	0.3383359	0.7000000	0.0133203
19.0500000	0.3885430	0.7500000	0.0152970
20.3200000	0.4422546	0.8000000	0.0174116
21.5900000	0.4994790	0.8500000	0.0196645
22.8600000	0.5602249	0.9000000	0.0220561
24.1300000	0.6245019	0.9500000	0.0245867
25.4000000	0.6923198	1.0000000	0.0272567
26.6700000	0.7636890	1.0500000	0.0300665
27.9400000	0.8386204	1.1000000	0.0330166
29.2100000	0.9171256	1.1500000	0.0361073
30.4800000	0.9992164	1.2000000	0.0393392
31.7500000	1.0849053	1.2500000	0.0427128
33.0200000	1.1742053	1.3000000	0.0462286
34.2900000	1.2671299	1.3500000	0.0498870
35.5600000	1.3636930	1.4000000	0.0536887
36.8300000	1.4639091	1.4500000	0.0576342

LDT NASA TELESCOPE ELEMENT 001 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
38.1000000	1.5677931	1.5000000	0.0617241
39.3700000	1.6753606	1.5500000	0.0659591
40.6400000	1.7866275	1.6000000	0.0703397
41.9100000	1.9016101	1.6500000	0.0748665
43.1800000	2.0203255	1.7000000	0.0795404
44.4500000	2.1427910	1.7500000	0.0843618
45.7200000	2.2690245	1.8000000	0.0893317
46.9900000	2.3990443	1.8500000	0.0944506
48.2600000	2.5328694	1.9000000	0.0997193
49.5300000	2.6705190	1.9500000	0.1051385
50.8000000	2.8120130	2.0000000	0.1107092
52.0700000	2.9573715	2.0500000	0.1164320
53.3400000	3.1066154	2.1000000	0.1223077
54.6100000	3.2597659	2.1500000	0.1283372
55.8800000	3.4168445	2.2000000	0.1345214
57.1500000	3.5778736	2.2500000	0.1408612
58.4200000	3.7428757	2.3000000	0.1473573
59.6900000	3.9118738	2.3500000	0.1540108
60.9600000	4.0848917	2.4000000	0.1608225
62.2300000	4.2619532	2.4500000	0.1677934
63.5000000	4.4430829	2.5000000	0.1749245
64.7700000	4.6283059	2.5500000	0.1822168
66.0400000	4.8176474	2.6000000	0.1896712
67.3100000	5.0111336	2.6500000	0.1972887
68.5800000	5.2087907	2.7000000	0.2050705
69.8500000	5.4106457	2.7500000	0.2130175
71.1200000	5.6167260	2.8000000	0.2211309
72.3900000	5.8270594	2.8500000	0.2294118
73.6600000	6.0416742	2.9000000	0.2378612
74.9300000	6.2605993	2.9500000	0.2464803
76.2000000	6.4838641	3.0000000	0.2552702
77.4700000	6.7114983	3.0500000	0.2642322
78.7400000	6.9435324	3.1000000	0.2733674
80.0100000	7.1799971	3.1500000	0.2826771
81.2800000	7.4209238	3.2000000	0.2921624
82.5500000	7.6663445	3.2500000	0.3018246
83.8200000	7.9162915	3.3000000	0.3116650
85.0900000	8.1707978	3.3500000	0.3216850
86.3600000	8.4298968	3.4000000	0.3318857
87.6300000	8.6936227	3.4500000	0.3422686
88.9000000	8.9620098	3.5000000	0.3528350
90.1700000	9.2350935	3.5500000	0.3635864
91.4400000	9.5129094	3.6000000	0.3745240
92.7100000	9.7954937	3.6500000	0.3856494
93.9800000	10.0828834	3.7000000	0.3969639
95.2500000	10.3751158	3.7500000	0.4084691
96.5200000	10.6722290	3.8000000	0.4201665

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 001 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
97.7900000	10.9742615	3.8500000	0.4320575
99.0600000	11.2812528	3.9000000	0.4441438
100.3300000	11.5932425	3.9500000	0.4564269
101.6000000	11.9102712	4.0000000	0.4689083
102.8700000	12.2323800	4.0500000	0.4815898
104.1400000	12.5596106	4.1000000	0.4944729
105.4100000	12.8920055	4.1500000	0.5075593
106.6800000	13.2296077	4.2000000	0.5208507
107.9500000	13.5724609	4.2500000	0.5343489
109.2200000	13.9206095	4.3000000	0.5480555
110.4900000	14.2740986	4.3500000	0.5619724
111.7600000	14.6329739	4.4000000	0.5761013
113.0300000	14.9972818	4.4500000	0.5904442
114.3000000	15.3670695	4.5000000	0.6050027
115.5700000	15.7423847	4.5500000	0.6197789
116.8400000	16.1232761	4.6000000	0.6347746
118.1100000	16.5097928	4.6500000	0.6499918
119.3800000	16.9019848	4.7000000	0.6654325
120.6500000	17.2999027	4.7500000	0.6810985
121.9200000	17.7035978	4.8000000	0.6969920
123.1900000	18.1131224	4.8500000	0.7131151
124.4600000	18.5285290	4.9000000	0.7294696
125.7300000	18.9498714	4.9500000	0.7460579
127.0000000	19.3772036	5.0000000	0.7628820
128.2700000	19.8105806	5.0500000	0.7799441
129.5400000	20.2500581	5.1000000	0.7972464
130.8100000	20.6956923	5.1500000	0.8147910
132.0800000	21.1475403	5.2000000	0.8325803
133.3500000	21.6056597	5.2500000	0.8506165
134.6200000	22.0701091	5.3000000	0.8689019
135.8900000	22.5409475	5.3500000	0.8874389
137.1600000	23.0182344	5.4000000	0.9062297
138.4300000	23.5020304	5.4500000	0.9252768
139.7000000	23.9923965	5.5000000	0.9445825
140.9700000	24.4893941	5.5500000	0.9641494
142.2400000	24.9930855	5.6000000	0.9839797
143.5100000	25.5035334	5.6500000	1.0040761
144.7800000	26.0208013	5.7000000	1.0244410
146.0500000	26.5449529	5.7500000	1.0450769
147.3200000	27.0760527	5.8000000	1.0659863
148.5900000	27.6141656	5.8500000	1.0871719
149.8600000	28.1593568	5.9000000	1.1086361
151.1300000	28.7116922	5.9500000	1.1303816
152.4000000	29.2712379	6.0000000	1.1524109
153.6700000	29.8380606	6.0500000	1.1747268
154.9400000	30.4122272	6.1000000	1.1973318
156.2100000	30.9938049	6.1500000	1.2202285

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

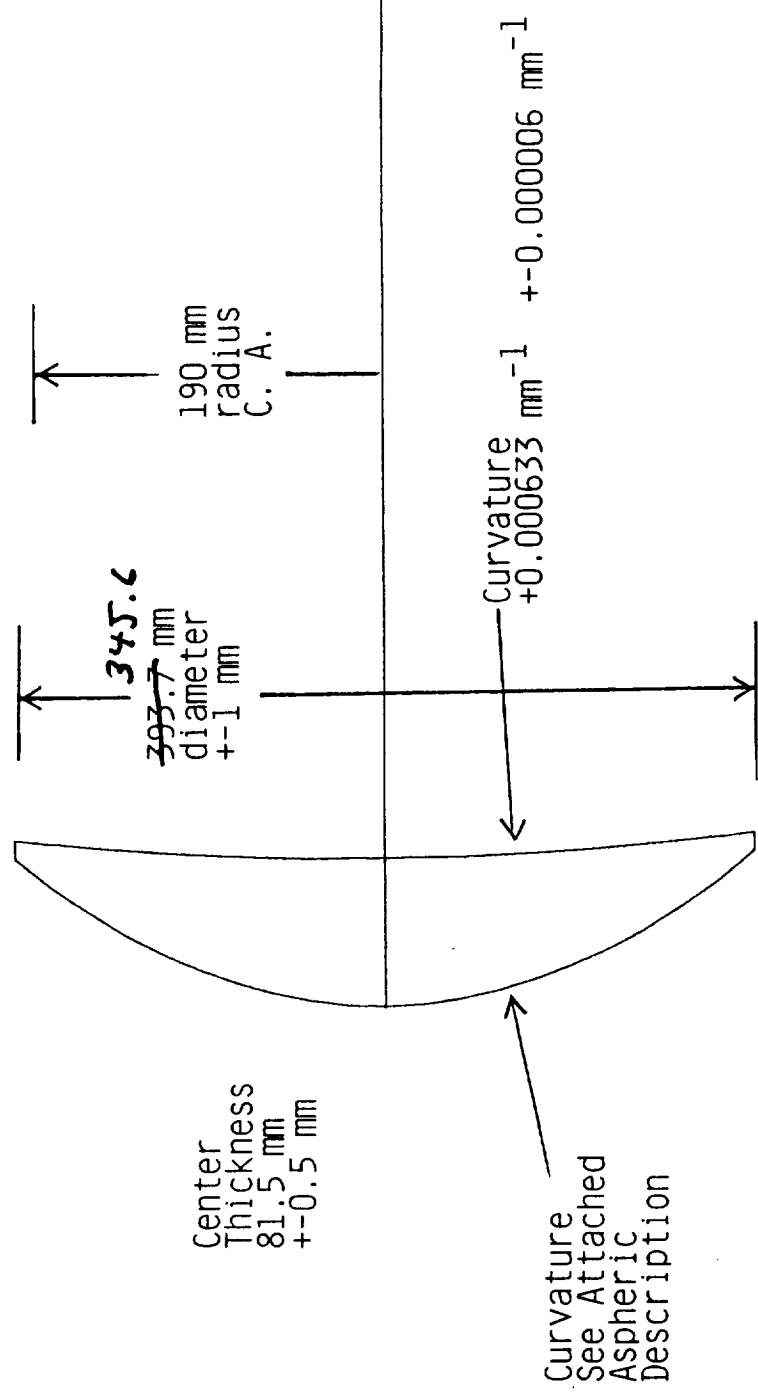
LDT NASA TELESCOPE ELEMENT 001 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
157.4800000	31.5828613	6.2000000	1.2434197
158.7500000	32.1794643	6.2500000	1.2669080
160.0200000	32.7836820	6.3000000	1.2906961
161.2900000	33.3955826	6.3500000	1.3147867
162.5600000	34.0152346	6.4000000	1.3391825
163.8300000	34.6427067	6.4500000	1.3638861
165.1000000	35.2780678	6.5000000	1.3889003
166.3700000	35.9213866	6.5500000	1.4142278
167.6400000	36.5727323	6.6000000	1.4398714
168.9100000	37.2321740	6.6500000	1.4658336
170.1800000	37.8997810	6.7000000	1.4921174
171.4500000	38.5756226	6.7500000	1.5187253
172.7200000	39.2597682	6.8000000	1.5456602
173.9900000	39.9522875	6.8500000	1.5729247
175.2600000	40.6532503	6.9000000	1.6005217
176.5300000	41.3627265	6.9500000	1.6284538
177.8000000	42.0807865	7.0000000	1.6567239
179.0700000	42.8075008	7.0500000	1.6853347
180.3400000	43.5429407	7.1000000	1.7142890
181.6100000	44.2871778	7.1500000	1.7435897
182.8800000	45.0402847	7.2000000	1.7732396
184.1500000	45.8023347	7.2500000	1.8032415
185.4200000	46.5734024	7.3000000	1.8335985
186.6900000	47.3535638	7.3500000	1.8643135
187.9600000	48.1428967	7.4000000	1.8953896
189.2300000	48.9414810	7.4500000	1.9268300
190.5000000	49.7493992	7.5000000	1.9586378
191.7700000	50.5667367	7.5500000	1.9908164
193.0400000	51.3935826	7.6000000	2.0233694
194.3100000	52.2300305	7.6500000	2.0563004
195.5800000	53.0761789	7.7000000	2.0896133
196.8500000	53.9321325	7.7500000	2.1233123

LDT NASA TELESCOPE ELEMENT 002

12-04-1990

WLP 7/23/91



Center Thickness
81.5 mm
 ± 0.5 mm

Curvature
See Attached
Aspheric
Description

PERSPECTIVE DISPLAYS, INC.
Mani L. Patel
Sheet 1 of 5

Material: Optical Grade
Methyl Methacrylate
(Acrylic)

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 002 SURFACE 1
Sag. vs. radial distance from optical axis

M.L.Pund 12/04/90
Perspective Displays,
Inc.

$$Z = (C*(R**2))/(1+\text{sqrt}(1-(1+K)*(C**2)*(R**2))) + A4*(R**4) + A6*(R**6) + A8*(R**8) + A10*(R**10)$$

C =	4.3240000E-03	/mm	Radius of Curvature
K =	-1.2475900E+00		Conic constant
A4 =	1.9018980E-11		Fourth order aspheric coefficient
A6 =	7.8000490E-14		Sixth order aspheric coefficient
A8 =	-1.6354920E-18		Eighth order aspheric coefficient
A10 =	2.5101750E-23		Tenth order aspheric coefficient

R = radial distance from optical axis (mm)

Z = Sag. (mm)

R (mm)	Z (mm)	R (inches)	Z (inches)
0.0000000	0.0000000	0.0000000	0.0000000
1.2700000	0.0034871	0.0500000	0.0001373
2.5400000	0.0139483	0.1000000	0.0005491
3.8100000	0.0313833	0.1500000	0.0012356
5.0800000	0.0557918	0.2000000	0.0021965
6.3500000	0.0871732	0.2500000	0.0034320
7.6200000	0.1255269	0.3000000	0.0049420
8.8900000	0.1708519	0.3500000	0.0067265
10.1600000	0.2231474	0.4000000	0.0087853
11.4300000	0.2824121	0.4500000	0.0111186
12.7000000	0.3486447	0.5000000	0.0137262
13.9700000	0.4218439	0.5500000	0.0166080
15.2400000	0.5020080	0.6000000	0.0197641
16.5100000	0.5891354	0.6500000	0.0231943
17.7800000	0.6832241	0.7000000	0.0268986
19.0500000	0.7842722	0.7500000	0.0308769
20.3200000	0.8922775	0.8000000	0.0351290
21.5900000	1.0072379	0.8500000	0.0396550
22.8600000	1.1291508	0.9000000	0.0444548
24.1300000	1.2580140	0.9500000	0.0495281
25.4000000	1.3938246	1.0000000	0.0548750
26.6700000	1.5365801	1.0500000	0.0604953
27.9400000	1.6862775	1.1000000	0.0663889
29.2100000	1.8429140	1.1500000	0.0725557
30.4800000	2.0064866	1.2000000	0.0789955
31.7500000	2.1769920	1.2500000	0.0857083
33.0200000	2.3544272	1.3000000	0.0926940
34.2900000	2.5387887	1.3500000	0.0999523
35.5600000	2.7300733	1.4000000	0.1074832
36.8300000	2.9282775	1.4500000	0.1152865

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 002 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
38.1000000	3.1333977	1.5000000	0.1233621
39.3700000	3.3454305	1.5500000	0.1317099
40.6400000	3.5643722	1.6000000	0.1403296
41.9100000	3.7902190	1.6500000	0.1492212
43.1800000	4.0229674	1.7000000	0.1583845
44.4500000	4.2626135	1.7500000	0.1678194
45.7200000	4.5091536	1.8000000	0.1775257
46.9900000	4.7625838	1.8500000	0.1875033
48.2600000	5.0229004	1.9000000	0.1977520
49.5300000	5.2900993	1.9500000	0.2082716
50.8000000	5.5641769	2.0000000	0.2190621
52.0700000	5.8451293	2.0500000	0.2301232
53.3400000	6.1329526	2.1000000	0.2414548
54.6100000	6.4276429	2.1500000	0.2530568
55.8800000	6.7291965	2.2000000	0.2649290
57.1500000	7.0376095	2.2500000	0.2770712
58.4200000	7.3528782	2.3000000	0.2894834
59.6900000	7.6749988	2.3500000	0.3021653
60.9600000	8.0039678	2.4000000	0.3151168
62.2300000	8.3397813	2.4500000	0.3283378
63.5000000	8.6824359	2.5000000	0.3418282
64.7700000	9.0319279	2.5500000	0.3555877
66.0400000	9.3882540	2.6000000	0.3696163
67.3100000	9.7514108	2.6500000	0.3839138
68.5800000	10.1213949	2.7000000	0.3984801
69.8500000	10.4982031	2.7500000	0.4133151
71.1200000	10.8818322	2.8000000	0.4284186
72.3900000	11.2722793	2.8500000	0.4437905
73.6600000	11.6695413	2.9000000	0.4594308
74.9300000	12.0736154	2.9500000	0.4753392
76.2000000	12.4844989	3.0000000	0.4915157
77.4700000	12.9021892	3.0500000	0.5079602
78.7400000	13.3266838	3.1000000	0.5246726
80.0100000	13.7579803	3.1500000	0.5416528
81.2800000	14.1960764	3.2000000	0.5589006
82.5500000	14.6409701	3.2500000	0.5764161
83.8200000	15.0926594	3.3000000	0.5941992
85.0900000	15.5511426	3.3500000	0.6122497
86.3600000	16.0164179	3.4000000	0.6305676
87.6300000	16.4884839	3.4500000	0.6491529
88.9000000	16.9673392	3.5000000	0.6680055
90.1700000	17.4529827	3.5500000	0.6871253
91.4400000	17.9454135	3.6000000	0.7065123
92.7100000	18.4446306	3.6500000	0.7261666
93.9800000	18.9506336	3.7000000	0.7460879
95.2500000	19.4634220	3.7500000	0.7662765
96.5200000	19.9829955	3.8000000	0.7867321

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 002 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
97.7900000	20.5093543	3.8500000	0.8074549
99.0600000	21.0424985	3.9000000	0.8284448
100.3300000	21.5824286	3.9500000	0.8497019
101.6000000	22.1291451	4.0000000	0.8712262
102.8700000	22.6826492	4.0500000	0.8930177
104.1400000	23.2429418	4.1000000	0.9150764
105.4100000	23.8100244	4.1500000	0.9374025
106.6800000	24.3838987	4.2000000	0.9599960
107.9500000	24.9645667	4.2500000	0.9828570
109.2200000	25.5520306	4.3000000	1.0059855
110.4900000	26.1462930	4.3500000	1.0293816
111.7600000	26.7473567	4.4000000	1.0530455
113.0300000	27.3552250	4.4500000	1.0769774
114.3000000	27.9699013	4.5000000	1.1011772
115.5700000	28.5913895	4.5500000	1.1256453
116.8400000	29.2196940	4.6000000	1.1503817
118.1100000	29.8548193	4.6500000	1.1753866
119.3800000	30.4967706	4.7000000	1.2006603
120.6500000	31.1455533	4.7500000	1.2262029
121.9200000	31.8011733	4.8000000	1.2520147
123.1900000	32.4636372	4.8500000	1.2780960
124.4600000	33.1329518	4.9000000	1.3044469
125.7300000	33.8091245	4.9500000	1.3310679
127.0000000	34.4921634	5.0000000	1.3579592
128.2700000	35.1820771	5.0500000	1.3851211
129.5400000	35.8788747	5.1000000	1.4125541
130.8100000	36.5825660	5.1500000	1.4402585
132.0800000	37.2931617	5.2000000	1.4682347
133.3500000	38.0106730	5.2500000	1.4964832
134.6200000	38.7351120	5.3000000	1.5250044
135.8900000	39.4664913	5.3500000	1.5537989
137.1600000	40.2048249	5.4000000	1.5828671
138.4300000	40.9501272	5.4500000	1.6122097
139.7000000	41.7024138	5.5000000	1.6418273
140.9700000	42.4617014	5.5500000	1.6717205
142.2400000	43.2280075	5.6000000	1.7018901
143.5100000	44.0013511	5.6500000	1.7323367
144.7800000	44.7817520	5.7000000	1.7630611
146.0500000	45.5692317	5.7500000	1.7940642
147.3200000	46.3638127	5.8000000	1.8253470
148.5900000	47.1655192	5.8500000	1.8569102
149.8600000	47.9743767	5.9000000	1.8887550
151.1300000	48.7904124	5.9500000	1.9208824
152.4000000	49.6136553	6.0000000	1.9532935
153.6700000	50.4441361	6.0500000	1.9859896
154.9400000	51.2818873	6.1000000	2.0189719
156.2100000	52.1269437	6.1500000	2.0522419

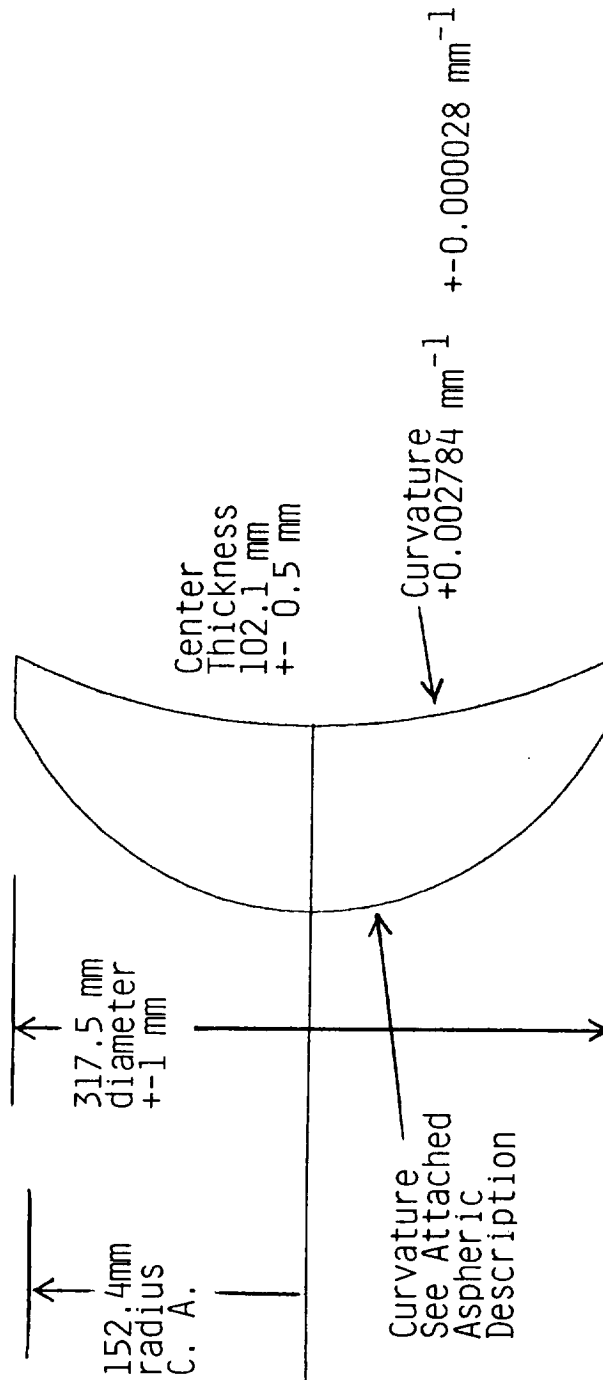
MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 002 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
157.4800000	52.9793419	6.2000000	2.0858009
158.7500000	53.8391209	6.2500000	2.1196504
160.0200000	54.7063223	6.3000000	2.1537922
161.2900000	55.5809897	6.3500000	2.1882279
162.5600000	56.4631699	6.4000000	2.2229594
163.8300000	57.3529122	6.4500000	2.2579887
165.1000000	58.2502689	6.5000000	2.2933177
166.3700000	59.1552955	6.5500000	2.3289486
167.6400000	60.0680506	6.6000000	2.3648839
168.9100000	60.9885966	6.6500000	2.4011259
170.1800000	61.9169994	6.7000000	2.4376771
171.4500000	62.8533286	6.7500000	2.4745405
172.7200000	63.7976581	6.8000000	2.5117188
173.9900000	64.7500661	6.8500000	2.5492152
175.2600000	65.7106351	6.9000000	2.5870329
176.5300000	66.6794526	6.9500000	2.6251753
177.8000000	67.6566109	7.0000000	2.6636461
179.0700000	68.6422077	7.0500000	2.7024491
180.3400000	69.6363460	7.1000000	2.7415884
181.6100000	70.6391350	7.1500000	2.7810683
182.8800000	71.6506897	7.2000000	2.8208933
184.1500000	72.6711315	7.2500000	2.8610682
185.4200000	73.7005886	7.3000000	2.9015980
186.6900000	74.7391965	7.3500000	2.9424880
187.9600000	75.7870976	7.4000000	2.9837440
189.2300000	76.8444424	7.4500000	3.0253717
190.5000000	77.9113897	7.5000000	3.0673775
191.7700000	78.9881064	7.5500000	3.1097680
193.0400000	80.0747686	7.6000000	3.1525499
194.3100000	81.1715618	7.6500000	3.1957308
195.5800000	82.2786812	7.7000000	3.2393182
196.8500000	83.3963323	7.7500000	3.2833202

12-04-1990

LDT NASA TELESCOPE ELEMENT 003



-A35-

PERSPECTIVE DISPLAYS, INC.

Material: Optical Grade
Methyl Methacrylate
(Acrylic)

Marvin L. Reed

Sheet 1 of 5

File ldtplas

SCALE = 0.25

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 003 SURFACE 1
Sag. vs. radial distance from optical axis

M.L.Pund 12/04/90
Perspective Displays,
Inc.

$$Z = (C*(R**2))/(1+\text{sqrt}(1-(1+K)*(C**2)*(R**2))) + A4*(R**4) + A6*(R**6) + A8*(R**8) + A10*(R**10)$$

C = 6.5930000E-03 /mm Radius of Curvature
K = -3.5479300E-01 Conic constant
A4 = -3.3579450E-09 Fourth order aspheric coefficient
A6 = 1.2774210E-14 Sixth order aspheric coefficient
A8 = -2.2076200E-18 Eighth order aspheric coefficient
A10 = 3.7341230E-23 Tenth order aspheric coefficient

R = radial distance from optical axis (mm)
Z = Sag. (mm)

R (mm)	Z (mm)	R (inches)	Z (inches)
0.0000000	0.0000000	0.0000000	0.0000000
1.2700000	0.0053170	0.0500000	0.0002093
2.5400000	0.0212685	0.1000000	0.0008373
3.8100000	0.0478565	0.1500000	0.0018841
5.0800000	0.0850840	0.2000000	0.0033498
6.3500000	0.1329553	0.2500000	0.0052345
7.6200000	0.1914760	0.3000000	0.0075384
8.8900000	0.2606529	0.3500000	0.0102619
10.1600000	0.3404941	0.4000000	0.0134053
11.4300000	0.4310088	0.4500000	0.0169689
12.7000000	0.5322078	0.5000000	0.0209531
13.9700000	0.6441028	0.5500000	0.0253584
15.2400000	0.7667071	0.6000000	0.0301853
16.5100000	0.9000349	0.6500000	0.0354344
17.7800000	1.0441022	0.7000000	0.0411064
19.0500000	1.1989260	0.7500000	0.0472018
20.3200000	1.3645246	0.8000000	0.0537214
21.5900000	1.5409179	0.8500000	0.0606661
22.8600000	1.7281269	0.9000000	0.0680365
24.1300000	1.9261742	0.9500000	0.0758336
25.4000000	2.1350837	1.0000000	0.0840584
26.6700000	2.3548808	1.0500000	0.0927118
27.9400000	2.5855921	1.1000000	0.1017950
29.2100000	2.8272460	1.1500000	0.1113089
30.4800000	3.0798722	1.2000000	0.1212548
31.7500000	3.3435019	1.2500000	0.1316339
33.0200000	3.6181679	1.3000000	0.1424476
34.2900000	3.9039046	1.3500000	0.1536970
35.5600000	4.2007480	1.4000000	0.1653838
36.8300000	4.5087356	1.4500000	0.1775093

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 003 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
38.1000000	4.8279067	1.5000000	0.1900751
39.3700000	5.1583023	1.5500000	0.2030828
40.6400000	5.4999649	1.6000000	0.2165341
41.9100000	5.8529392	1.6500000	0.2304307
43.1800000	6.2172712	1.7000000	0.2447745
44.4500000	6.5930091	1.7500000	0.2595673
45.7200000	6.9802028	1.8000000	0.2748111
46.9900000	7.3789044	1.8500000	0.2905080
48.2600000	7.7891676	1.9000000	0.3066601
49.5300000	8.2110485	1.9500000	0.3232696
50.8000000	8.6446049	2.0000000	0.3403388
52.0700000	9.0898971	2.0500000	0.3578700
53.3400000	9.5469873	2.1000000	0.3758656
54.6100000	10.0159401	2.1500000	0.3943284
55.8800000	10.4968224	2.2000000	0.4132607
57.1500000	10.9897034	2.2500000	0.4326655
58.4200000	11.4946548	2.3000000	0.4525455
59.6900000	12.0117507	2.3500000	0.4729036
60.9600000	12.5410679	2.4000000	0.4937428
62.2300000	13.0826857	2.4500000	0.5150664
63.5000000	13.6366865	2.5000000	0.5368774
64.7700000	14.2031549	2.5500000	0.5591793
66.0400000	14.7821791	2.6000000	0.5819756
67.3100000	15.3738497	2.6500000	0.6052697
68.5800000	15.9782607	2.7000000	0.6290654
69.8500000	16.5955093	2.7500000	0.6533665
71.1200000	17.2256959	2.8000000	0.6781770
72.3900000	17.8689244	2.8500000	0.7035010
73.6600000	18.5253021	2.9000000	0.7293426
74.9300000	19.1949401	2.9500000	0.7557063
76.2000000	19.8779533	3.0000000	0.7825966
77.4700000	20.5744603	3.0500000	0.8100181
78.7400000	21.2845841	3.1000000	0.8379758
80.0100000	22.0084516	3.1500000	0.8664745
81.2800000	22.7461943	3.2000000	0.8955195
82.5500000	23.4979483	3.2500000	0.9251161
83.8200000	24.2638541	3.3000000	0.9552698
85.0900000	25.0440576	3.3500000	0.9859865
86.3600000	25.8387095	3.4000000	1.0172720
87.6300000	26.6479659	3.4500000	1.0491325
88.9000000	27.4719886	3.5000000	1.0815744
90.1700000	28.3109451	3.5500000	1.1146041
91.4400000	29.1650090	3.6000000	1.1482287
92.7100000	30.0343602	3.6500000	1.1824551
93.9800000	30.9191854	3.7000000	1.2172908
95.2500000	31.8196780	3.7500000	1.2527432
96.5200000	32.7360388	3.8000000	1.2888204

LDT NASA TELESCOPE ELEMENT 003 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
97.7900000	33.6684763	3.8500000	1.3255306
99.0600000	34.6172067	3.9000000	1.3628822
100.3300000	35.5824549	3.9500000	1.4008841
101.6000000	36.5644543	4.0000000	1.4395454
102.8700000	37.5634479	4.0500000	1.4788759
104.1400000	38.5796880	4.1000000	1.5188854
105.4100000	39.6134375	4.1500000	1.5595842
106.6800000	40.6649700	4.2000000	1.6009831
107.9500000	41.7345704	4.2500000	1.6430933
109.2200000	42.8225357	4.3000000	1.6859266
110.4900000	43.9291755	4.3500000	1.7294951
111.7600000	45.0548131	4.4000000	1.7738115
113.0300000	46.1997858	4.4500000	1.8188892
114.3000000	47.3644459	4.5000000	1.8647420
115.5700000	48.5491620	4.5500000	1.9113843
116.8400000	49.7543193	4.6000000	1.9588315
118.1100000	50.9803214	4.6500000	2.0070993
119.3800000	52.2275909	4.7000000	2.0562044
120.6500000	53.4965709	4.7500000	2.1061642
121.9200000	54.7877263	4.8000000	2.1569971
123.1900000	56.1015454	4.8500000	2.2087223
124.4600000	57.4385411	4.9000000	2.2613599
125.7300000	58.7992532	4.9500000	2.3149312
127.0000000	60.1842501	5.0000000	2.3694587
128.2700000	61.5941305	5.0500000	2.4249658
129.5400000	63.0295265	5.1000000	2.4814774
130.8100000	64.4911051	5.1500000	2.5390199
132.0800000	65.9795719	5.2000000	2.5976209
133.3500000	67.4956735	5.2500000	2.6573100
134.6200000	69.0402010	5.3000000	2.7181181
135.8900000	70.6139938	5.3500000	2.7800785
137.1600000	72.2179436	5.4000000	2.8432261
138.4300000	73.8529990	5.4500000	2.9075984
139.7000000	75.5201704	5.5000000	2.9732351
140.9700000	77.2205360	5.5500000	3.0401786
142.2400000	78.9552475	5.6000000	3.1084743
143.5100000	80.7255380	5.6500000	3.1781708
144.7800000	82.5327290	5.7000000	3.2493200
146.0500000	84.3782406	5.7500000	3.3219780
147.3200000	86.2636005	5.8000000	3.3962047
148.5900000	88.1904564	5.8500000	3.4720652
149.8600000	90.1605892	5.9000000	3.5496295
151.1300000	92.1759279	5.9500000	3.6289735
152.4000000	94.2385673	6.0000000	3.7101798
153.6700000	96.3507883	6.0500000	3.7933381
154.9400000	98.5150815	6.1000000	3.8785465
156.2100000	100.7341746	6.1500000	3.9659124

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 003 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
97.7900000	33.6684763	3.8500000	1.3255306
99.0600000	34.6172067	3.9000000	1.3628822
100.3300000	35.5824549	3.9500000	1.4008841
101.6000000	36.5644543	4.0000000	1.4395454
102.8700000	37.5634479	4.0500000	1.4788759
104.1400000	38.5796880	4.1000000	1.5188854
105.4100000	39.6134375	4.1500000	1.5595842
106.6800000	40.6649700	4.2000000	1.6009831
107.9500000	41.7345704	4.2500000	1.6430933
109.2200000	42.8225357	4.3000000	1.6859266
110.4900000	43.9291755	4.3500000	1.7294951
111.7600000	45.0548131	4.4000000	1.7738115
113.0300000	46.1997858	4.4500000	1.8188892
114.3000000	47.3644459	4.5000000	1.8647420
115.5700000	48.5491620	4.5500000	1.9113843
116.8400000	49.7543193	4.6000000	1.9588315
118.1100000	50.9803214	4.6500000	2.0070993
119.3800000	52.2275909	4.7000000	2.0562044
120.6500000	53.4965709	4.7500000	2.1061642
121.9200000	54.7877263	4.8000000	2.1569971
123.1900000	56.1015454	4.8500000	2.2087223
124.4600000	57.4385411	4.9000000	2.2613599
125.7300000	58.7992532	4.9500000	2.3149312
127.0000000	60.1842501	5.0000000	2.3694587
128.2700000	61.5941305	5.0500000	2.4249658
129.5400000	63.0295265	5.1000000	2.4814774
130.8100000	64.4911051	5.1500000	2.5390199
132.0800000	65.9795719	5.2000000	2.5976209
133.3500000	67.4956735	5.2500000	2.6573100
134.6200000	69.0402010	5.3000000	2.7181181
135.8900000	70.6139938	5.3500000	2.7800785
137.1600000	72.2179436	5.4000000	2.8432261
138.4300000	73.8529990	5.4500000	2.9075984
139.7000000	75.5201704	5.5000000	2.9732351
140.9700000	77.2205360	5.5500000	3.0401786
142.2400000	78.9552475	5.6000000	3.1084743
143.5100000	80.7255380	5.6500000	3.1781708
144.7800000	82.5327290	5.7000000	3.2493200
146.0500000	84.3782406	5.7500000	3.3219780
147.3200000	86.2636005	5.8000000	3.3962047
148.5900000	88.1904564	5.8500000	3.4720652
149.8600000	90.1605892	5.9000000	3.5496295
151.1300000	92.1759279	5.9500000	3.6289735
152.4000000	94.2385673	6.0000000	3.7101798
153.6700000	96.3507883	6.0500000	3.7933381
154.9400000	98.5150815	6.1000000	3.8785465
156.2100000	100.7341746	6.1500000	3.9659124

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

LDT NASA TELESCOPE ELEMENT 003 SURFACE 1

R (mm)	Z (mm)	R (inches)	Z (inches)
157.4800000	103.0110648	6.2000000	4.0555537
158.7500000	105.3490575	6.2500000	4.1476007

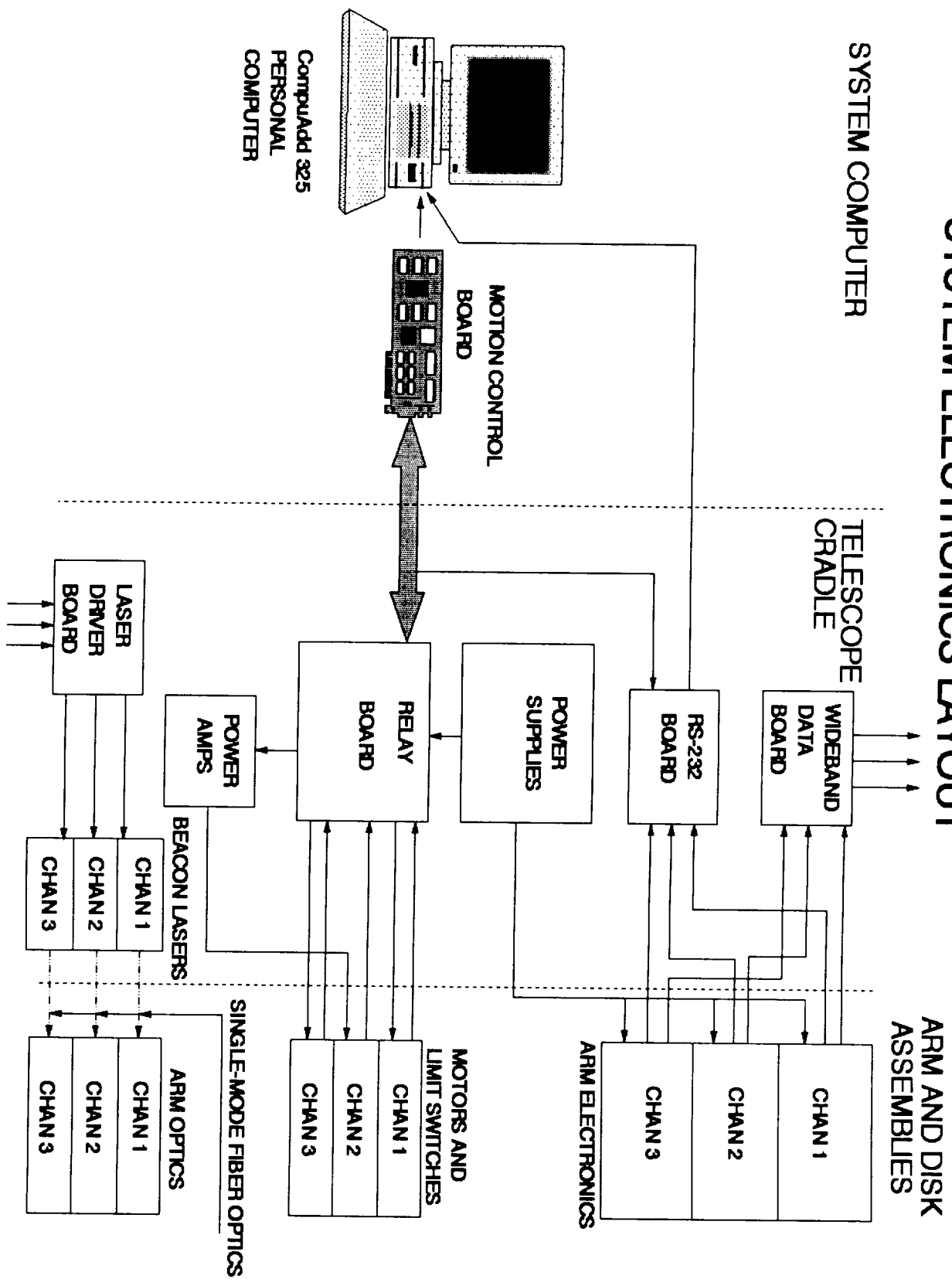
APPENDIX B

COMPLETE ELECTRONIC SCHEMATICS

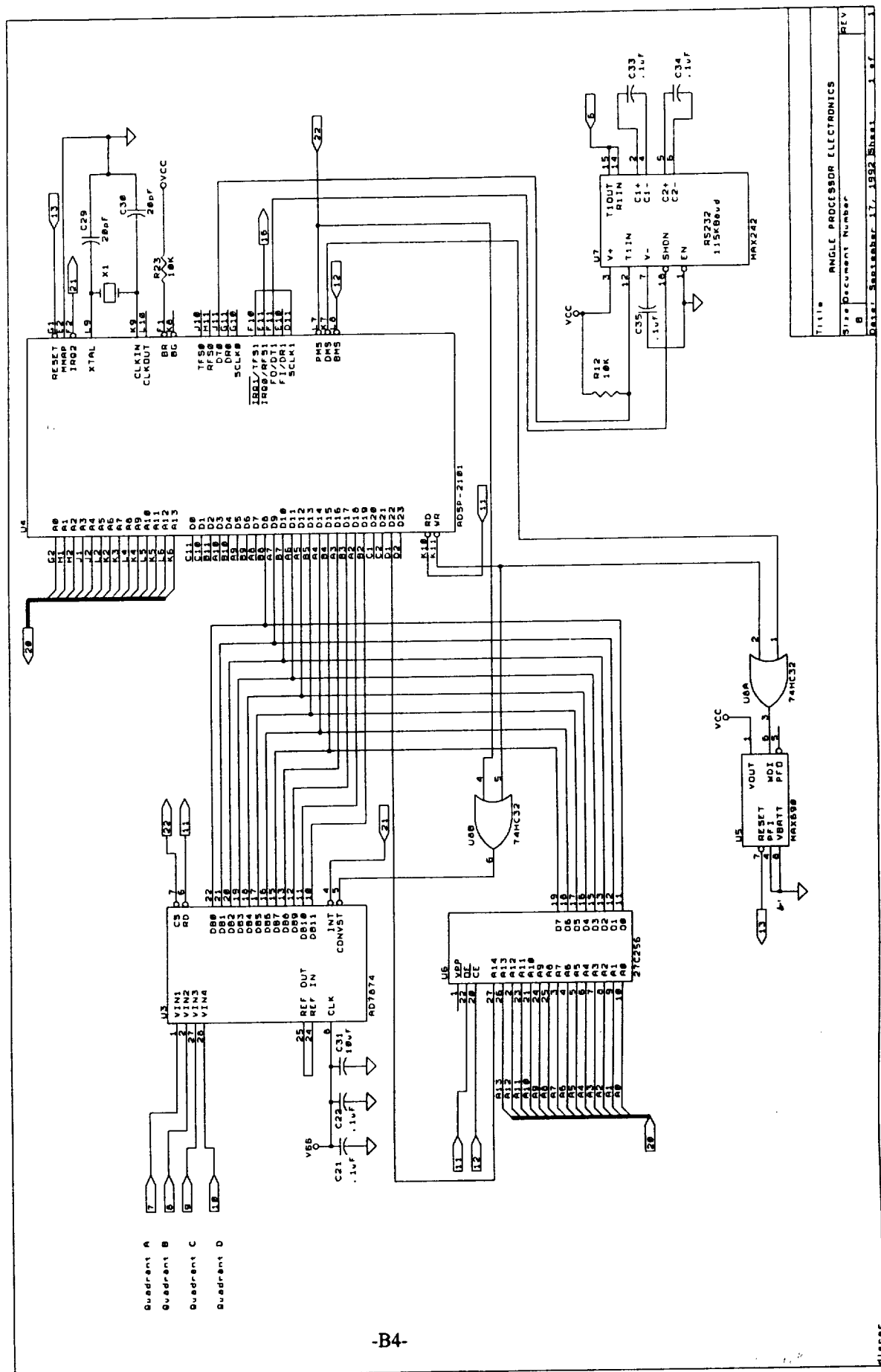
	Page
System Electronics Layout	B3
Angle Processor Electronics	B4
Quadrant Detector Electronics	B5
Wideband Amplifier	B6
Motor Switching (Ch 1)	B7
Motor Switching (Ch 2)	B8
Motor Switching (Ch 3)	B9
Relay Card Miscellaneous Circuit	B10
Ch 1 Cable from Relay Card to Disk Assy	B11
Ch 2 Cable from Relay Card to Disk Assy	B12
Ch 3 Cable from Relay Card to Disk Assy	B13
Receiver Electronics Interconnect	B14
Rec. Power Distribution to Custom Cable	B15
RS232 & Sync Pulse Interconnect Card	B16
Single Channel Driver Circuit (Ch 1, typical)	B17
Demonstration Test Aid	B18
Laser Diode Driver Card	B19
OMA 6-axis Motion Control Card (Sh 1-6)	B20-B25

THIS PAGE IS INTENTIONALLY LEFT BLANK

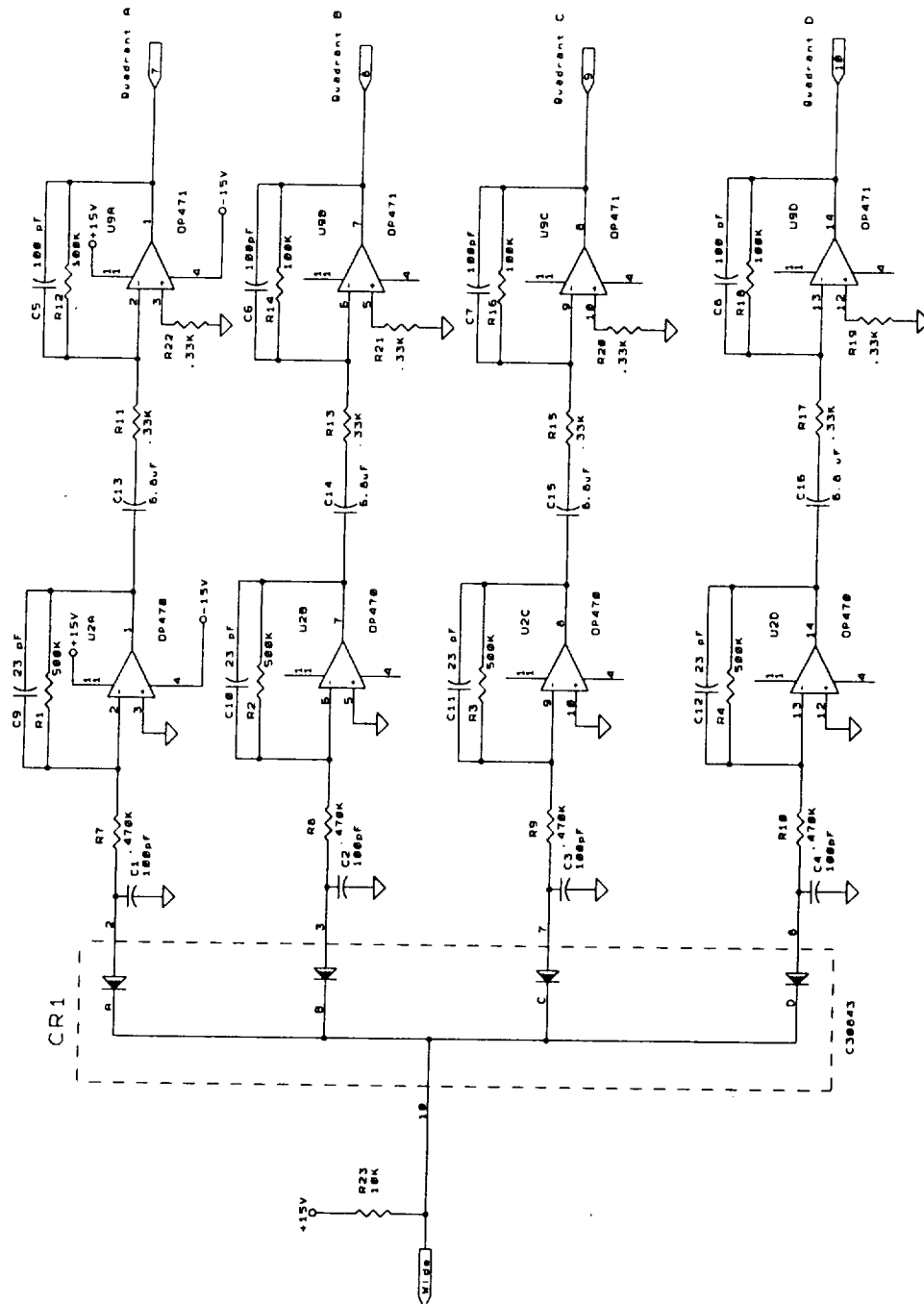
SYSTEM ELECTRONICS LAYOUT



MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

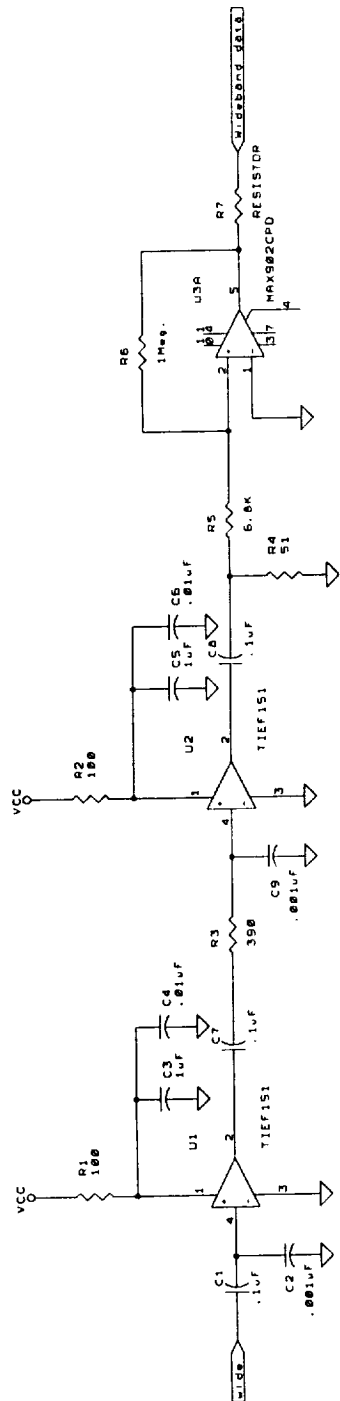


MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



Laser Data Technology	
Title	QUADRANT DETECTOR ELECTRONICS
Size	8
Document Number	REV
Date	November 3, 1992 Sheet 1 of 1

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



-B6-

Laser Data Technology, Inc.
9375 Dismen Industrial Drive
St. Louis, MO 63132

Title

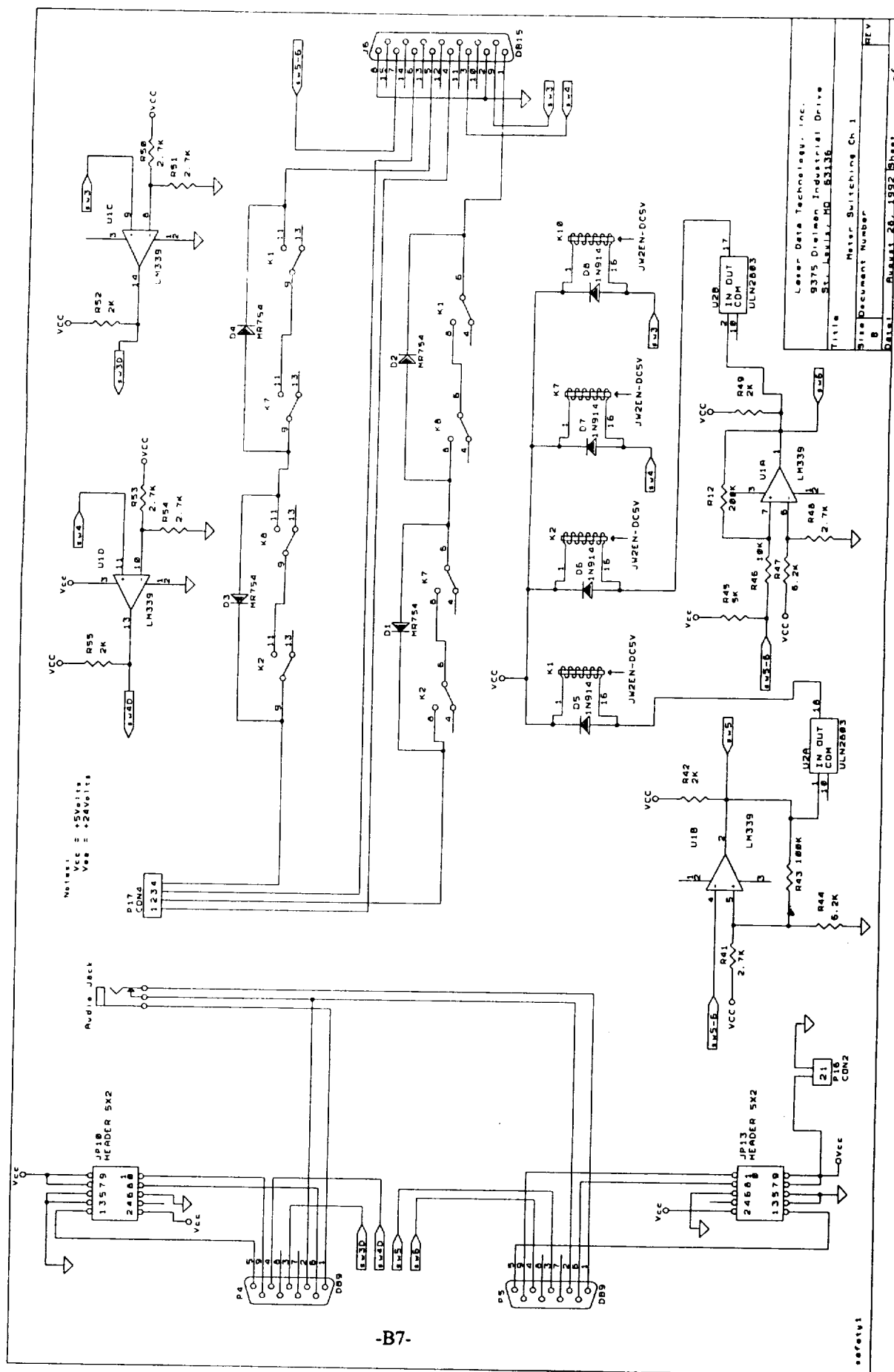
Wideband Amplifier

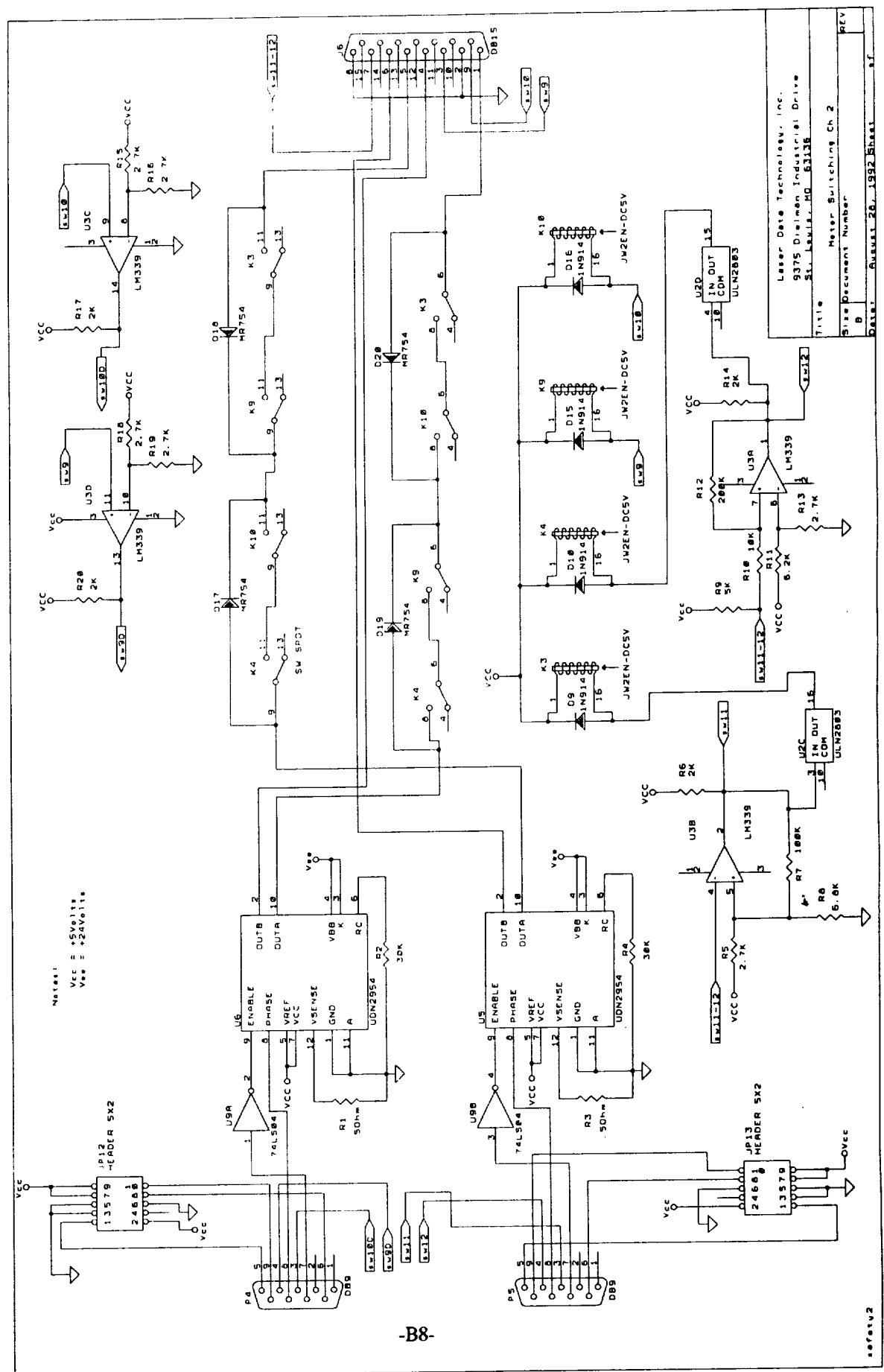
Size Document Number

REV

Date: August 23, 1992 Sheet 1 of 1

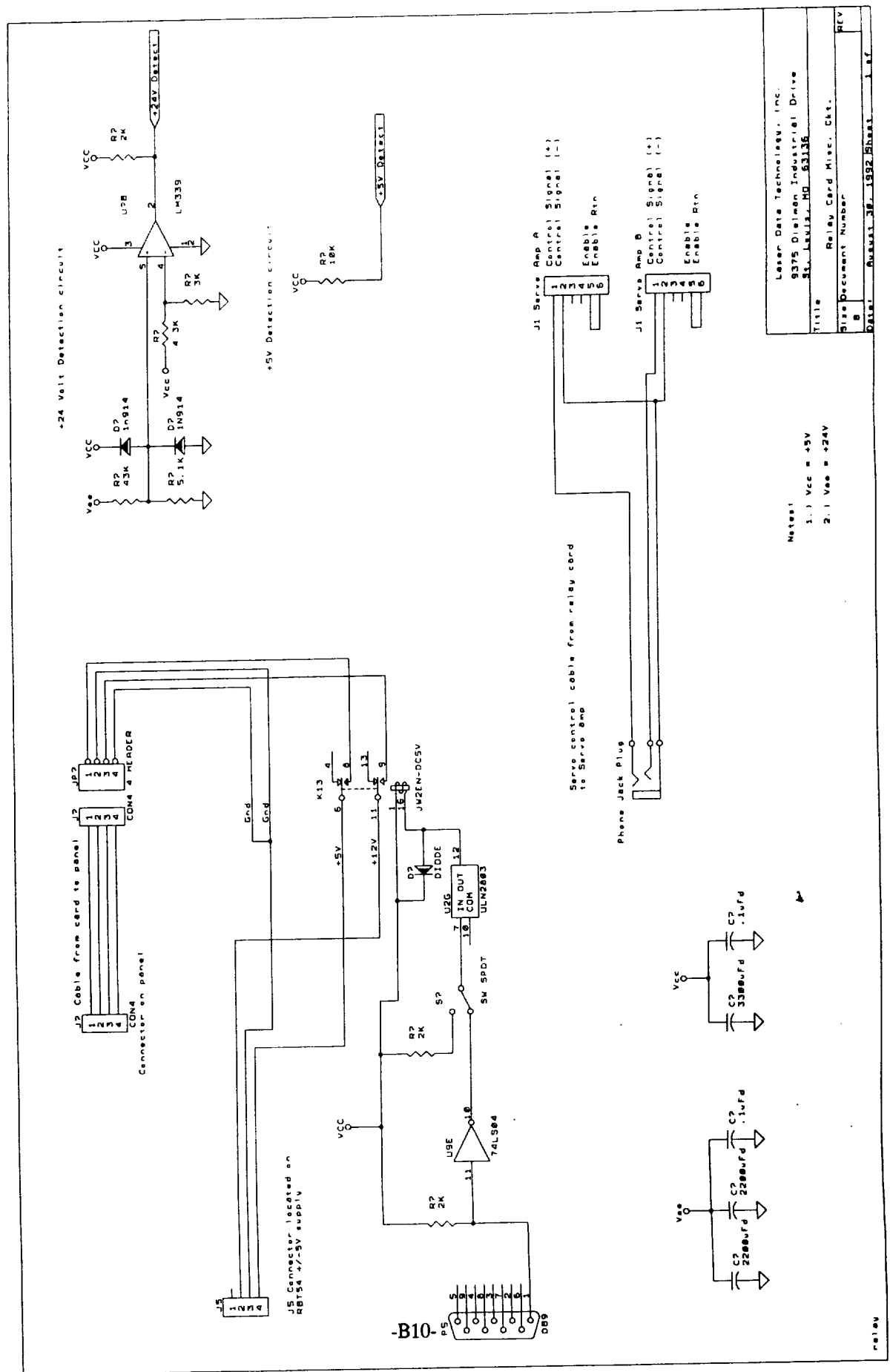
wide







MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT





Laser Data Technology, Inc.
9375 Dielman Industrial Drive
St. Louis, MO 63136

Title

Size	Part Number	Description
		Ch 1 Cable from Relay Card to Disk Assembly

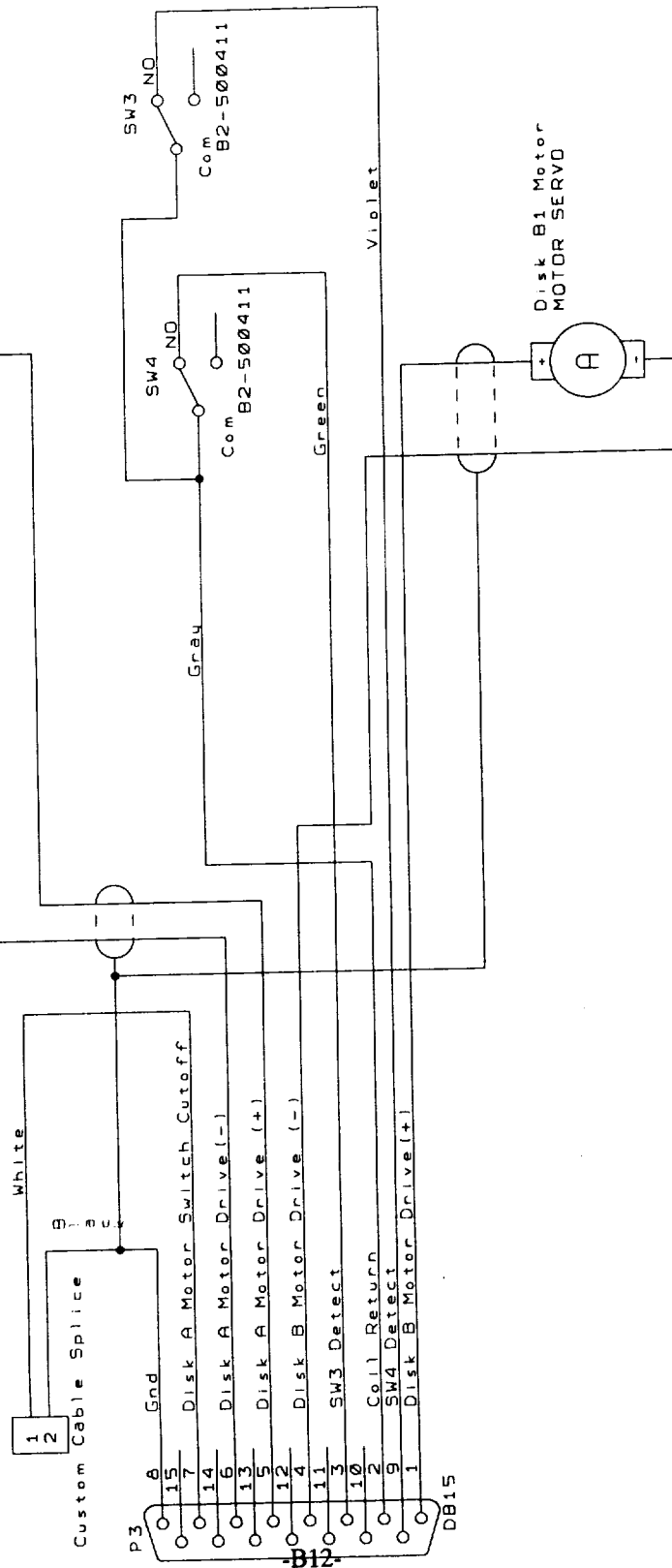
Size	Document Number
------	-----------------

Date: August 28, 1992 Sheet of

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT

Channel 2

Note: Pin 2 will splice into the shield of wire 7 in the custom cable & pin 1 will splice into wire 6 of the custom cable



Laser Data Technology, Inc.
9375 Dielman Industrial Drive
St. Louis, MO 63136

Title Ch 2 Cable Relay Card to Disk Assembly

Size Document Number

A

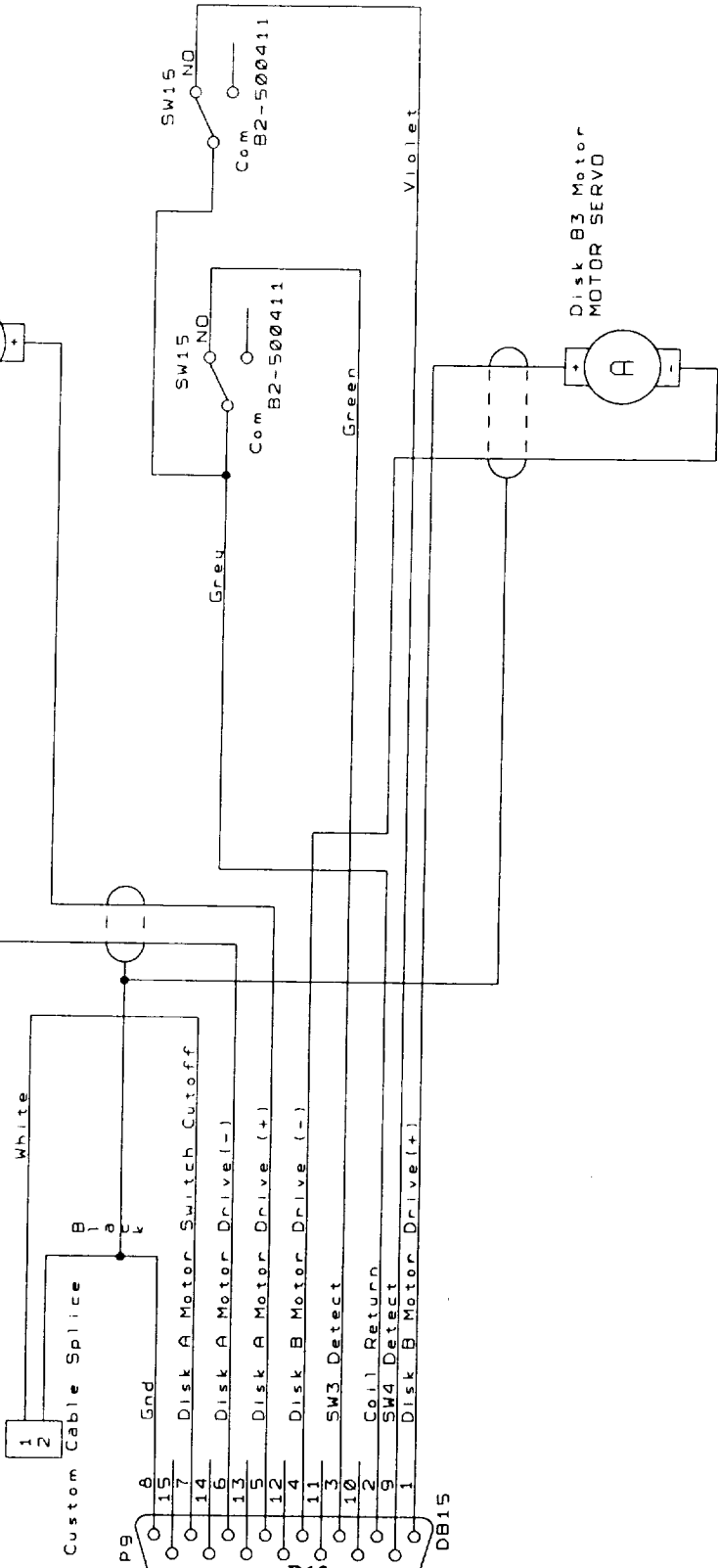
REV

Date: August 28, 1992 Sheet 1 of 1

Cable 1

Channel 3

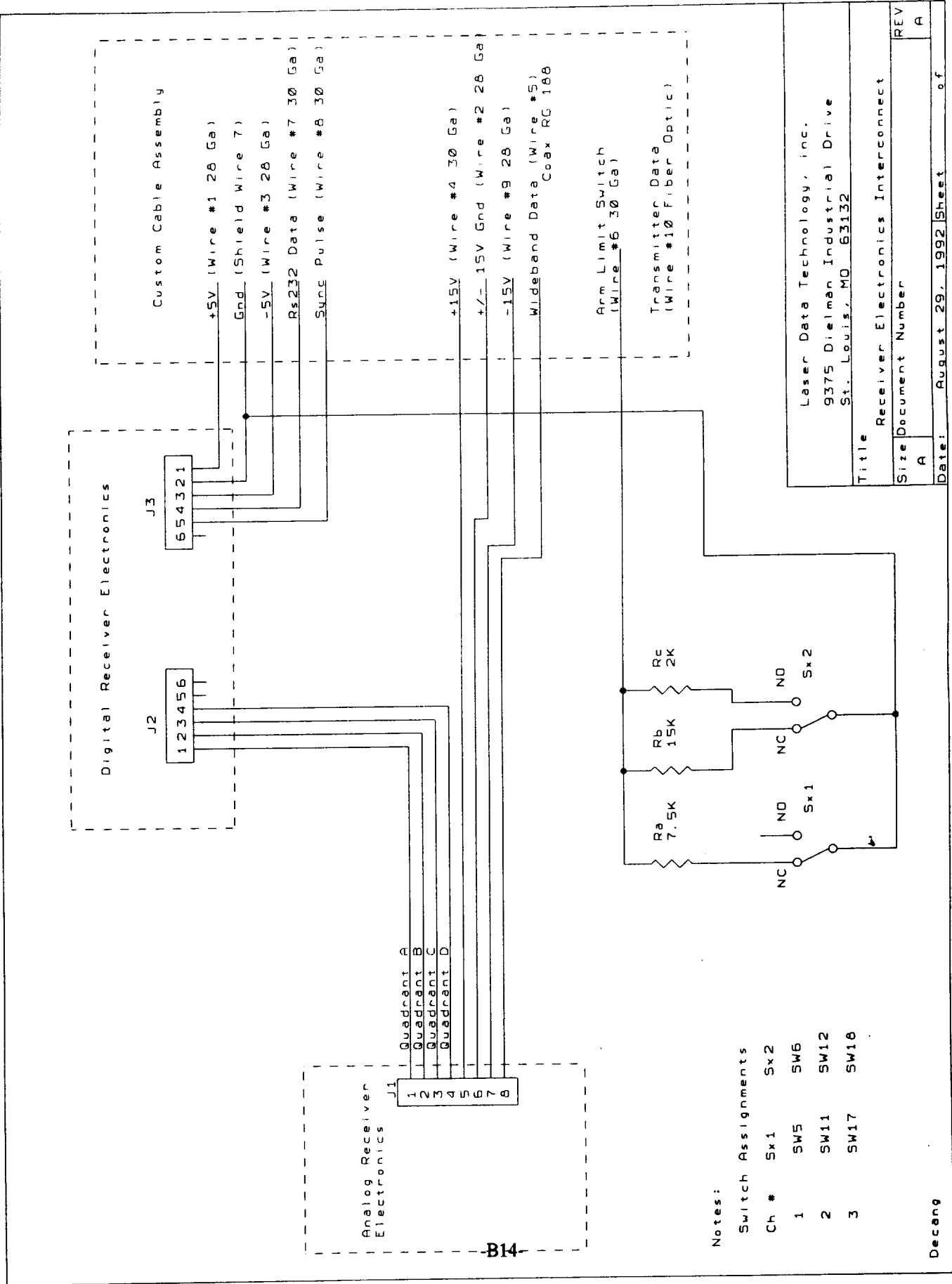
Note:
Pin 2 will splice into the
shield of wire 7 in the custom cable
& pin 1 will splice into wire 6 of the
custom cable



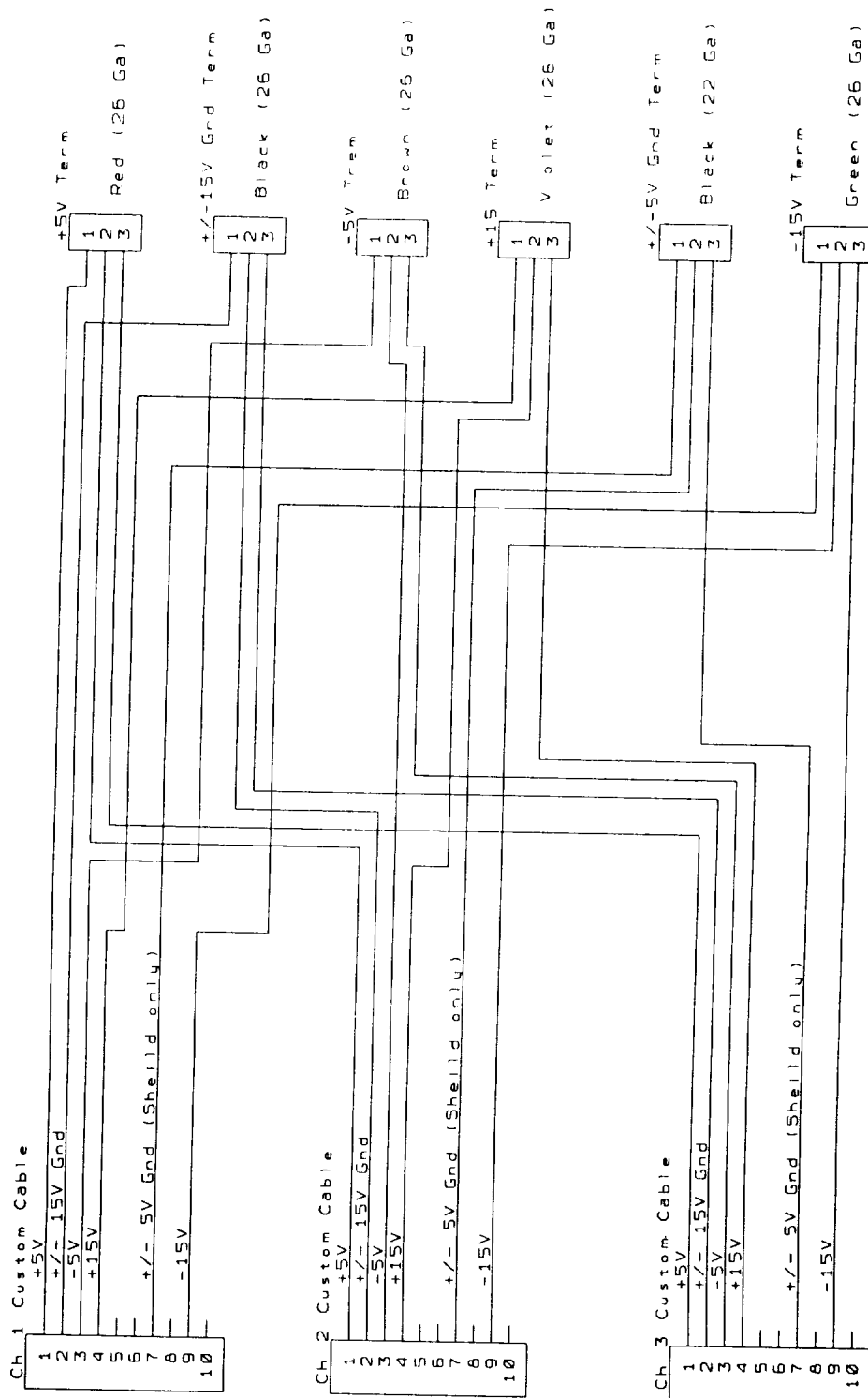
Laser Data Technology, Inc.
9375 Dielman Industrial Drive
St. Louis, MO 63136

Title		Ch 3 Cable Relay Card to Disk Assembly
Size	Document Number	REV
A		
Date:	August 28, 1992	Sheet 5 of 5

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



-B15-

Laser Data Technology, Inc.
9375 Dielman Industrial Drive
St. Louis, MO 63136

Title

Rec. Power Distribution to Custom Cable

Size Document Number

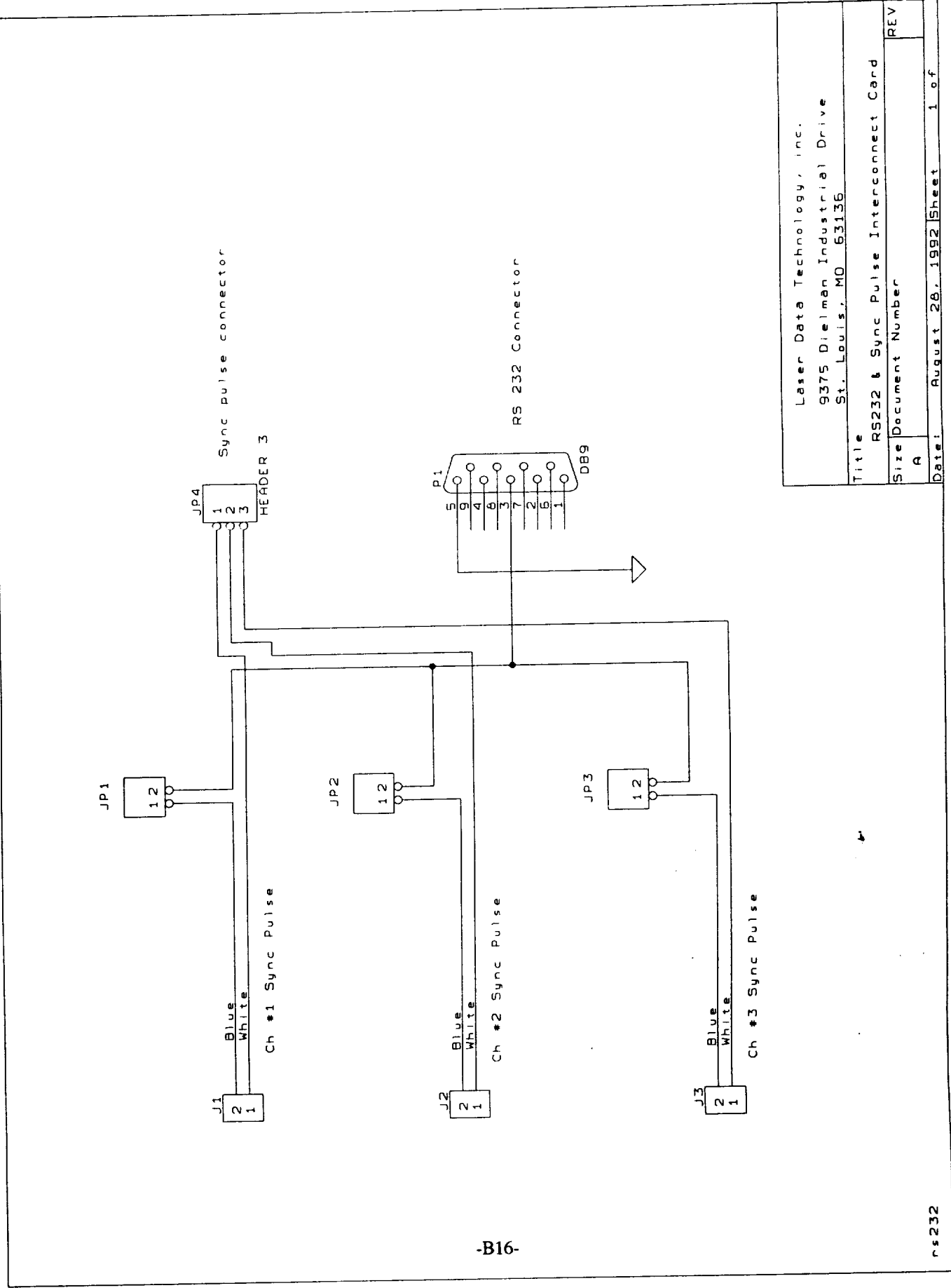
A

REV

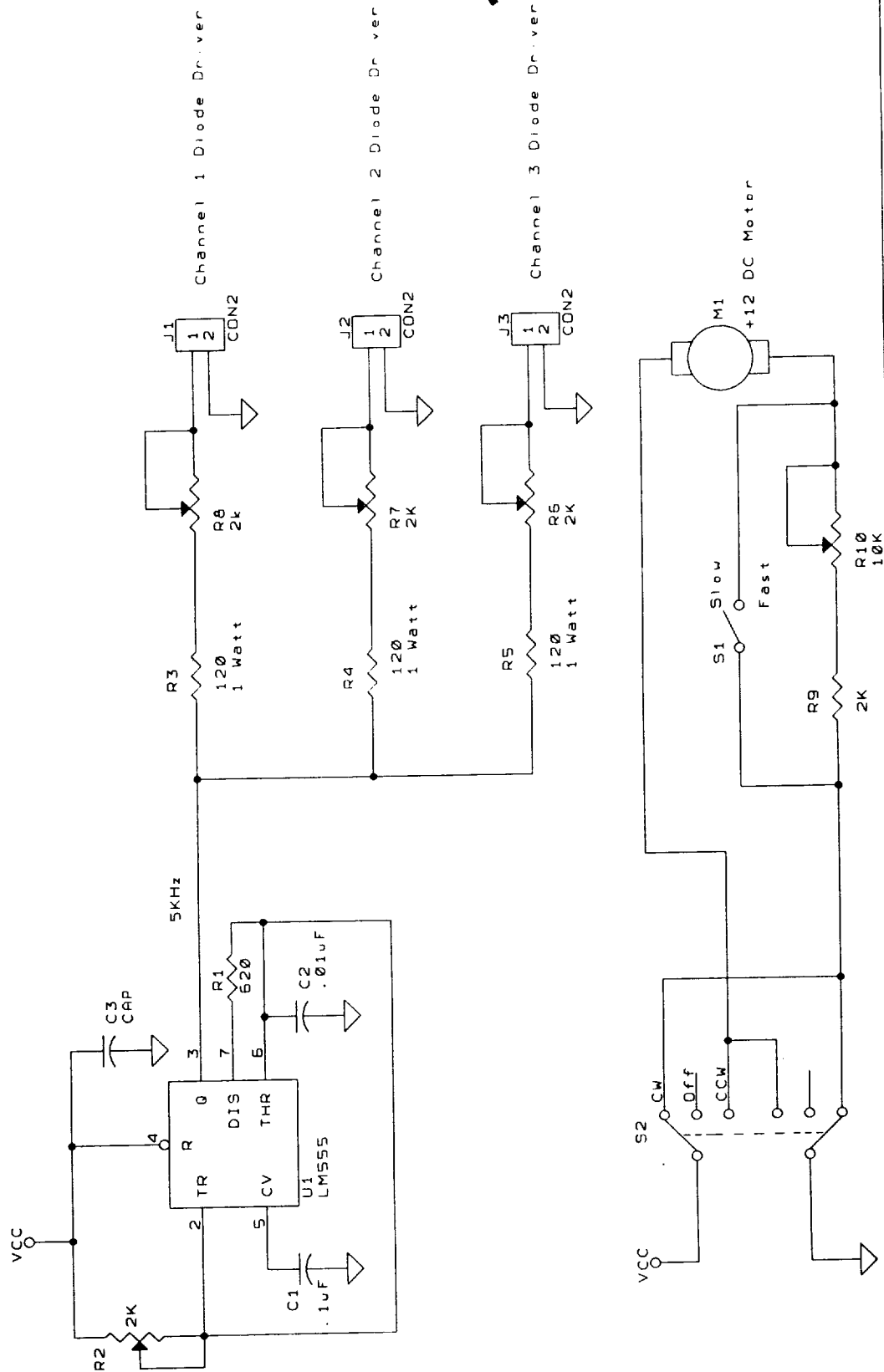
Date: August 28, 1992 Sheet 1 of 1

Power1

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



Laser Data Technology, Inc. 9375 Dielman Industrial Drive St. Louis, MO 63136	
Title	RS232 & Sync Pulse Interconnect Card
Size	Document Number
REV	
Date:	August 28, 1992 Sheet 1 of 1



-B18-

Notes:

- 1.) Vcc = +12V
- 2.) U1 pin 8 is Vcc and pin 4 is Gnd

Demolas

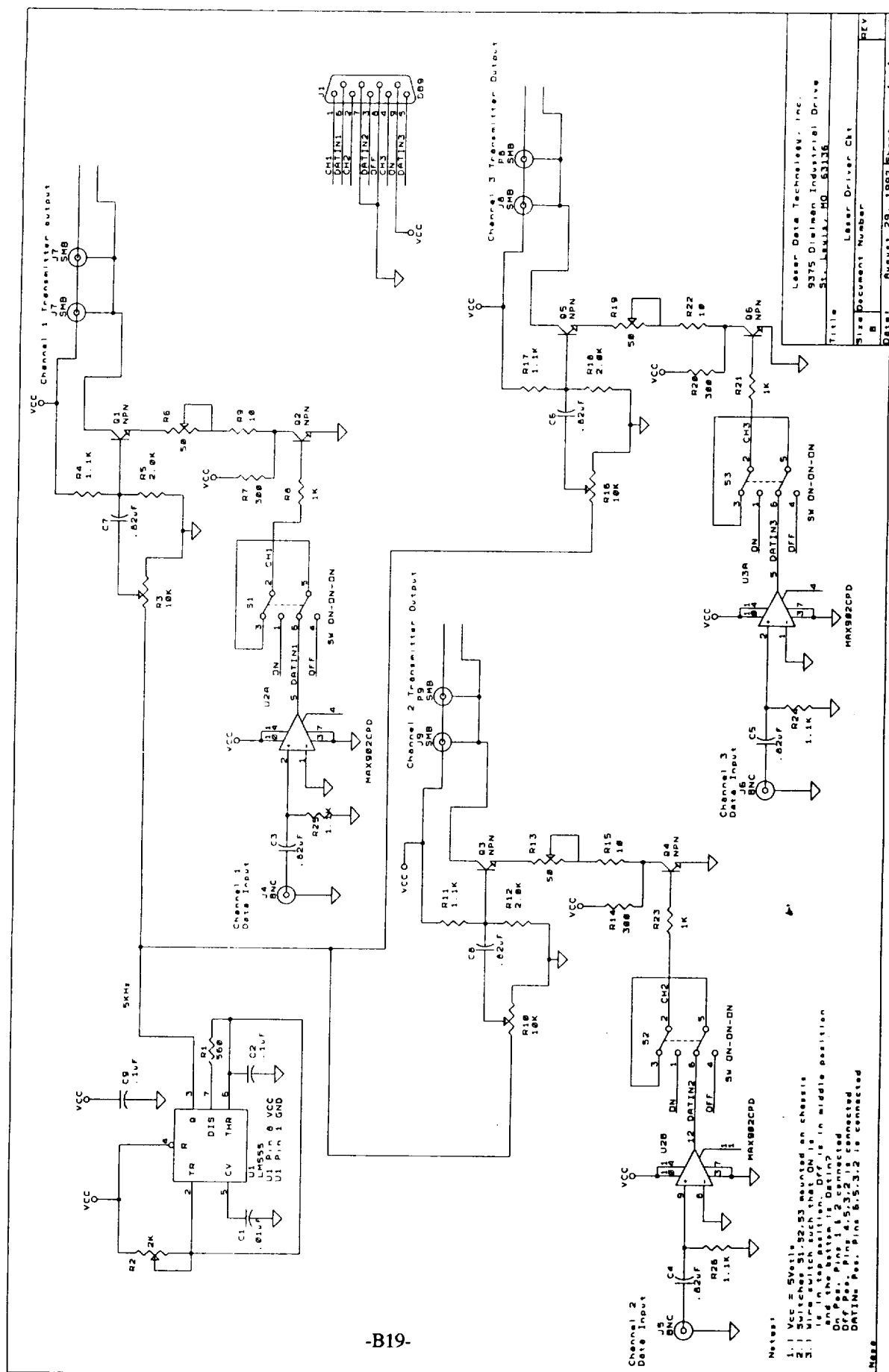
Laser Data Technology, Inc.
9375 Dielman Industrial Drive
St. Louis, MO 63136

Title Demonstration Test Aid

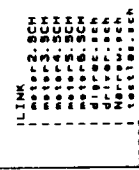
Size Document Number

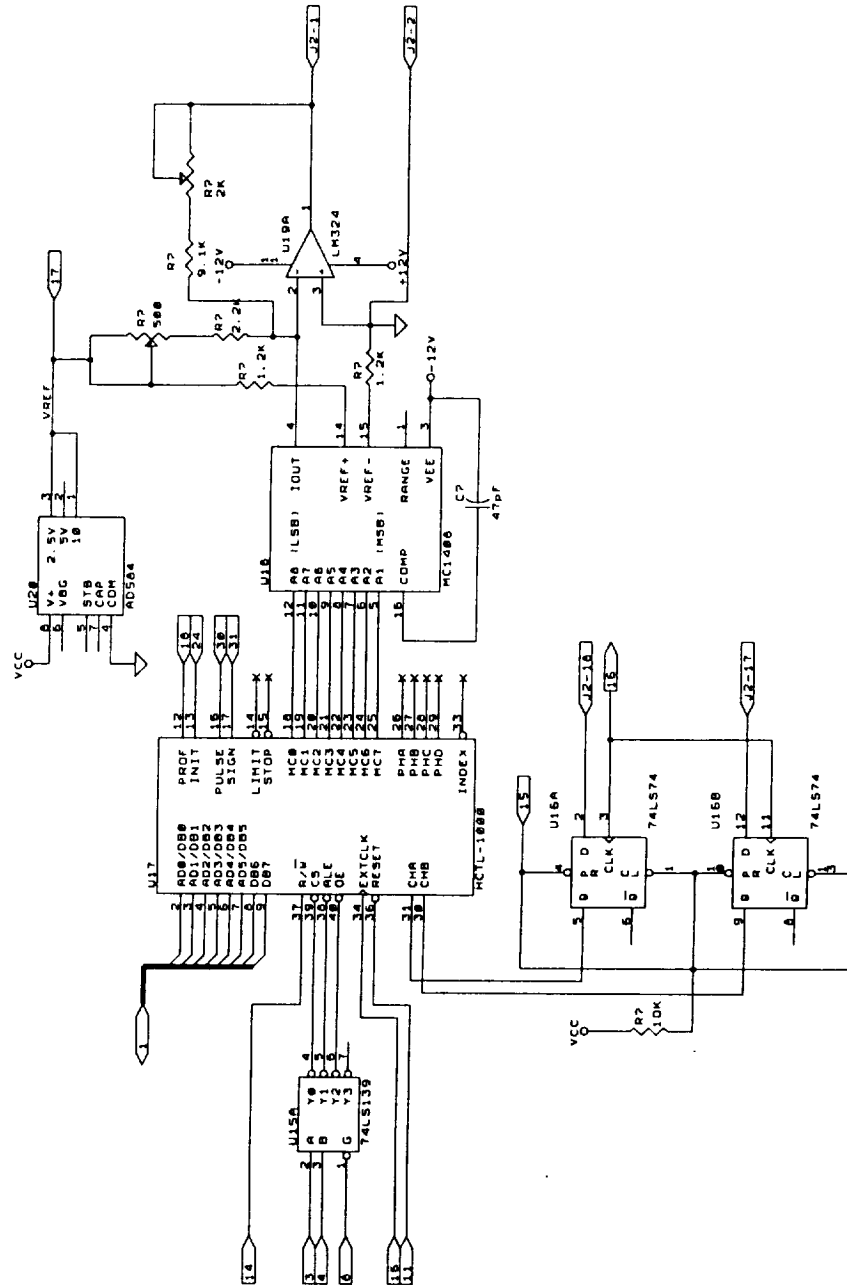
A

Date: August 29, 1992 Sheet 1 of 1



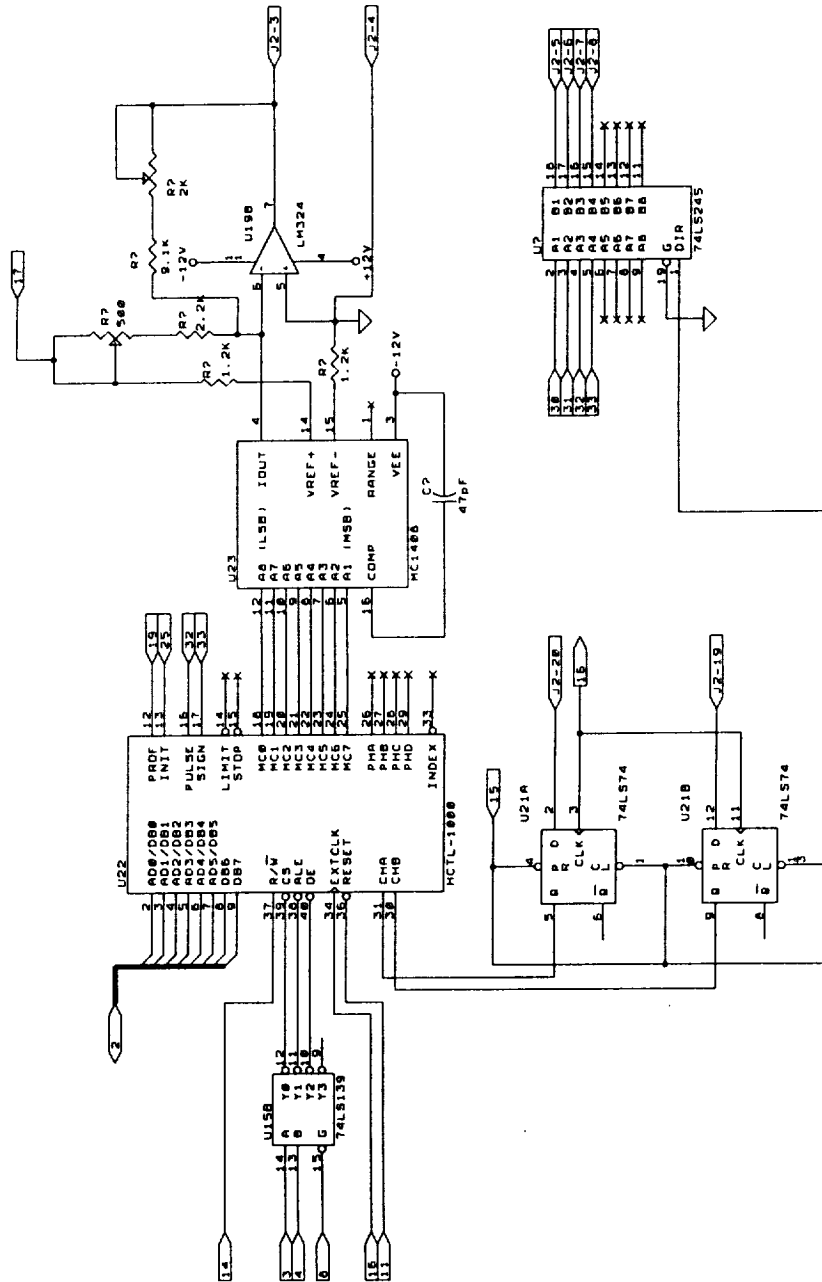
Laser Data Technology, Inc.		STV	
9375 Diehlman Industrial Drive		A	
St. Louis, MO 63132		6	
Title		Date	
DMA 6 AXIS MOTION CONTROL CARD		November 3, 1992	
B		Sheet 1 of 6	
Document Number		STV	





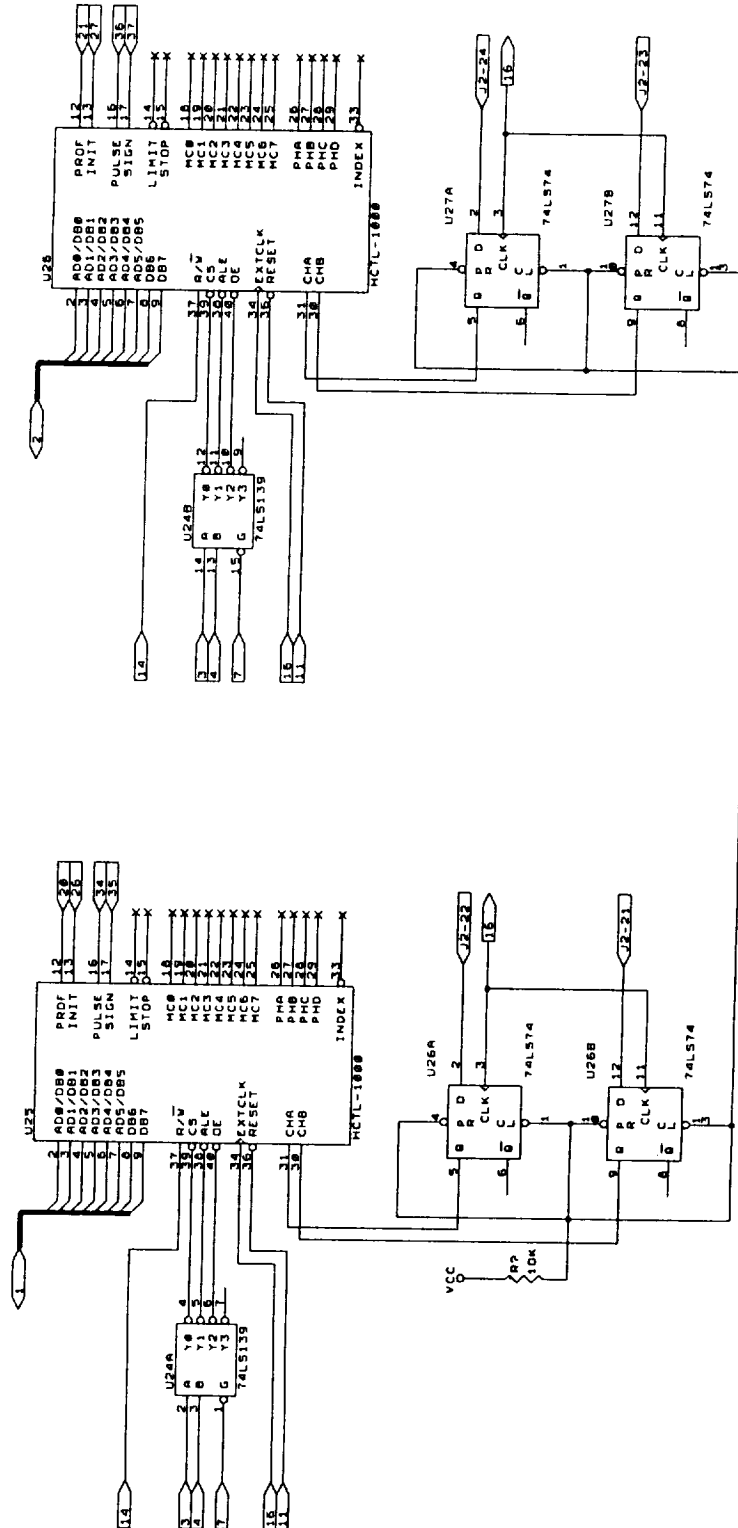
AXIS 2 MOTOR CONTROLLER PAIR

Laser Data Technology, Inc.	
9375 Dielman Industrial Drive	
St. Louis, MO 63132	
Title	OMA 6 AXIS MOTION CONTROL CARD
Size	Document Number
Rev	A
Date	November 1, 1992 Sheet 1 of 6



AXIS 2 MOTOR CONTROLLER PAIR

Laser Data Technology, Inc.	
9375 Dielman Industrial Drive	
St. Louis, MO 63132	
Title OMA 6 AXIS MOTION CONTROL CARD	
Size	8 1/2" x 11"
Document Number	100-100000-001
Date	March 1, 1992
Page	1 of 5

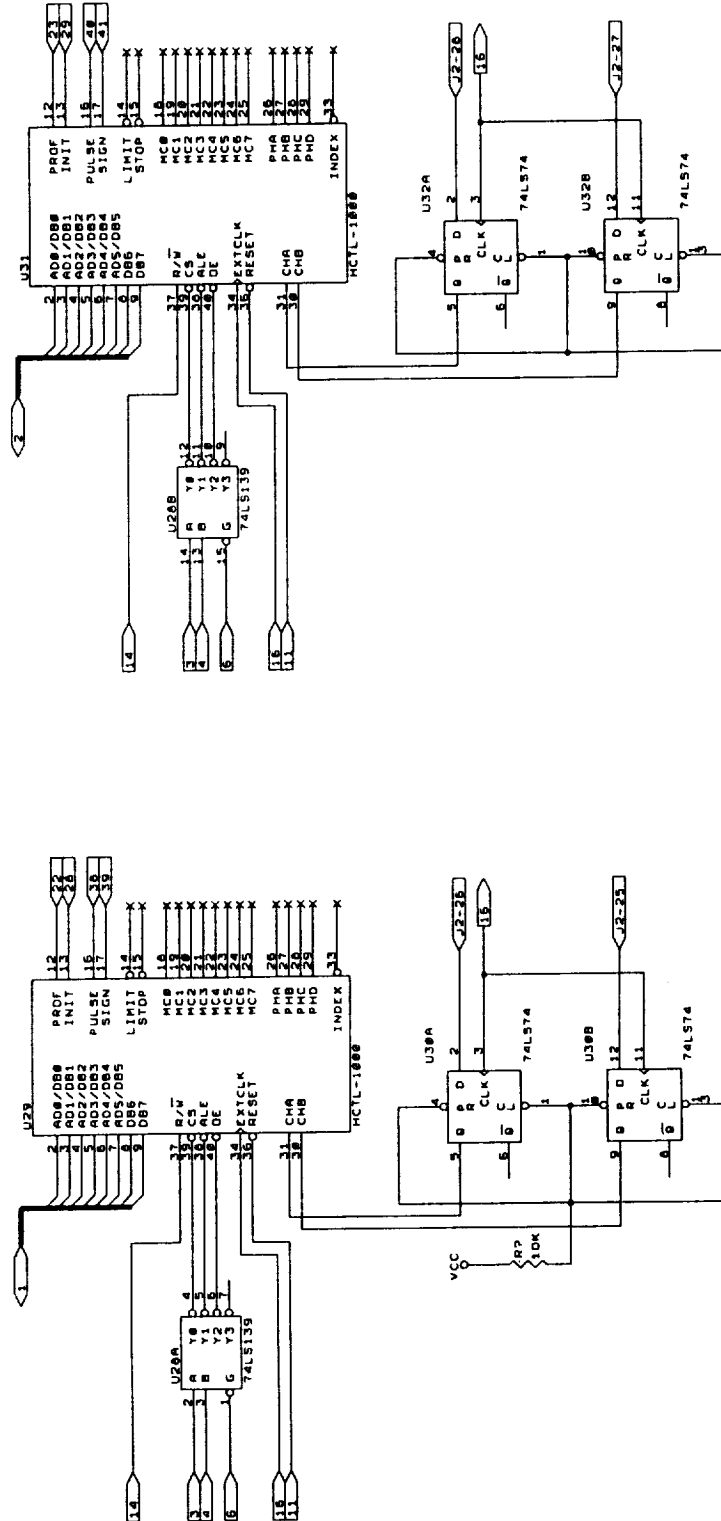


AXIS 1 MOTOR CONTROLLER PAIR

-B23-

Laser Data Technology, Inc.	
9375 Diehlman Industrial Drive	
St. Louis, MO 63132	
Title	OMA 6 AXIS MOTION CONTROL CARD
Size	Document Number
Date	November 1, 1992 Sheet 1 of 6

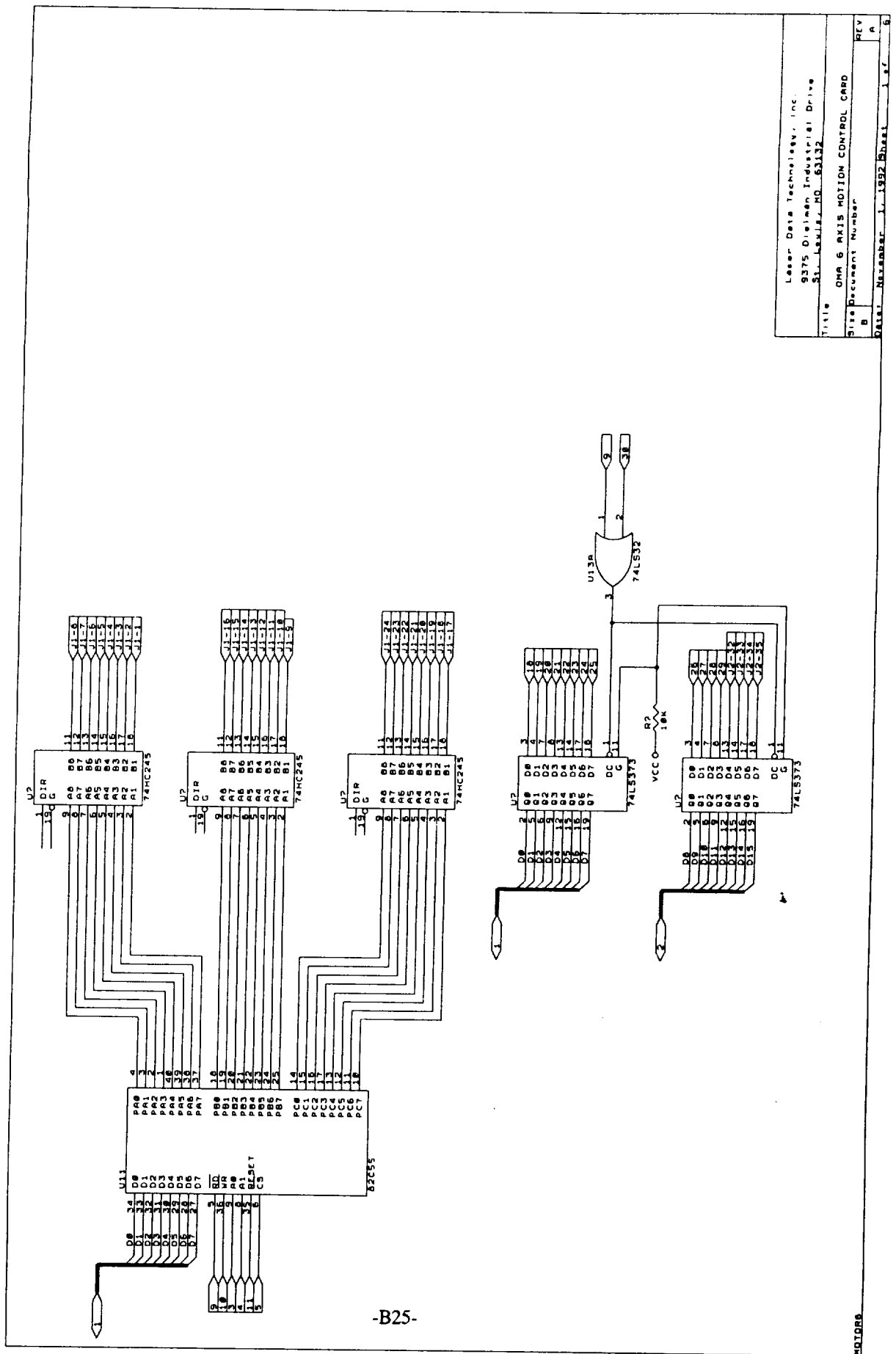
MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



AXIS 3 MOTOR CONTROLLER PAIR

Laser Data Technology, Inc.	
9375 Dielsien Industrial Drive	
St. Louis, MO 63132	
Title	QMA 6 AXIS MOTION CONTROL CARD
Drawn	Document Number
Rev	REV
Date	Number 1, 1992
Sheet	1 of 5

MULTI-ACCESS FREE SPACE LASER COMMUNICATIONS TERMINAL - FINAL REPORT



Laser Data Technology, Inc.
 9375 Dielman Industrial Drive
 St. Louis, MO 63132
 Title OMA 6 AXIS MOTION CONTROL CARD
 Size Document Number B
 Date November 1, 1992 Sheet 1 of 6

APPENDIX C

COMPLETE SOFTWARE LISTINGS

Program	Page
COMPUTE.C	C3
DISCRETE.C	C5
DISPLAY.C	C9
FILE.C	C25
INITMOT.C	C30
MANUAL.C	C34
MENU.C	C36
MONITOR.C	C42
MOTCNT.C	C47
MOTOR.C	C61
NEWMOT.C	C70
SERL.C	C78
TRACK.C	C82
AZEL.DAT	C91
INPUT.DAT	C92
ADSYS.DSP	C93
ALGORTHM.DSP	C95
BANDFILT.DSP	C99
LOWPASS.DSP	C102
MAIN.DSP	C104
UART.DSP	C112
ATTBLK.FCB	C117
CIRCBK.FCB	C118
DISCBK.FCB	C119
FOVBLK.FCB	C120
MISCBK.FCB	C121
ORBITBLK.FCB	C122
SAVEBLK.FCB	C123
ALDLDLAB.FOR	C124
ANGLEPR.FOR	C126
ARKTNS.FOR	C129
ASGNMFN.FOR	C130
AZELALDL.FOR	C134
BLKDATA.FOR	C135
CNTRLFN.FOR	C136
COPYVEC.FOR	C143
CROSS.FOR	C144
DFQ.FOR	C145
DISCPOS.FOR	C146
DOT.FOR	C149
ELSTAT.FOR	C150
GAUSCL.FOR	C151
GAUSS.FOR	C152
INTRFACE.FOR	C153

COMPLETE SOFTWARE LISTINGS (cont'd)

Program	Page
INV3X3.FOR	C159
NAVIGFN.FOR	C160
NAVIGPR.FOR	C162
OMA.FOR	C164
QFITVAL.FOR	C165
RNDM.FOR	C167
RUK.FOR	C168
SCANGN.FOR	C169
SCENEPR.FOR	C171
TRUTHGN.FOR	C174
UNIT.FOR	C176
VLEN.FOR	C177
XKEP.FOR	C178
XYAZEL.FOR	C179

```

1
2  /* Compute.C: Contains all programs for computing Azimuth and Elevation
3     numbers*/
4
5  #include <stdio.h>
6  #include <conio.h>
7  #include <ctype.h>
8  #include <time.h>
9  #include <graph.h>
10 #include <dos.h>
11 #include <time.h>
12 #include <math.h>
13 #include <float.h>
14
15 #define PI 3.141592654
16
17  /* Compute_Az_El computes azimuth and elevation parameters based on
18     motor encoder positions StepA and StepB Reflect the positions of the
19     of DiskA and DiskB*/
20
21  Compute_AZ_EL      (long int StepA,long int StepB,
22                      int Print_to_screen, double *Az_El, double *Alpha_Delta)
23  {
24      double Gear_ratio,Counts_per_degree;
25      double Tangle,alpha,delta,Theta;
26      double Pie_180 = 3.141592654/180;
27
28      /* 354 teeth per gear with 2000 motor encoder counts per one 360 degree
29         motor shaft rotation.  4.640 (in.) is the radius of the drive drum and
30         1.70(in) is the radius of the pivot point */
31
32      Counts_per_degree = (354.0/360.0)*2000.0;
33      Gear_ratio = Counts_per_degree /(4.640/1.70);
34
35      Tangle = 0.0;
36      Tangle = sin(1.211116e-05 * (fabs((double)StepB - (double)StepA)));
37
38      alpha = 90.0 + (((double)StepA - (double)StepB)/Gear_ratio);
39      Alpha_Delta[0] = alpha;
40      delta = ((double)StepA / Counts_per_degree);
41      Alpha_Delta[1] = delta;
42
43      if (alpha >= 90.0)
44          Theta = (3.141592654/2.0) + acos(Tangle) - (delta * Pie_180);
45      else
46          Theta = (3.141592654/2.0) - acos(Tangle) - (delta * Pie_180);
47
48      Az_El[0] = atan(0.5427 * Tangle * cos(Theta));      /* Compute AZ error */
49      Az_El[1] = atan(0.5427 * Tangle * sin(Theta));      /* Compute Elevation error */
50
51      if (Print_to_screen == 1)
52      {
53          printf("\n IstepA = %9li IstepB = %9li", StepA,StepB);
54          printf("\n Delta = %7.4f Deg. Alpha = %7.4f Deg.",delta,alpha);
55          printf("\n T = %10.8f Rad. Theta = %10.7f Rad.",Tangle,Theta);
56          printf("\n Azimuth = %10.8f Rad. Elevation = %10.8f Rad.",
57              Az_El[0],Az_El[1]);
58          printf("\n Azimuth = %10.7f Deg. Elevation = %10.7f Deg.",
59              Az_El[0]*57.29577951,Az_El[1]*57.29577951);
60          printf("\n\n");
61          Wait_for_key_Press();
62      }
63  }
64
65  /* Compute_Az_El computes azimuth and elevation parameters based on
66     X and Y position in millimeters */
67
68  Compute_Az_El_From_XY (double x, double y, int Print_results, double *AzEl)
69  {
70      {
71          AzEl[0] = atan((x/116.0) * 0.194380309);
72          AzEl[1] = atan((y/116.0) * 0.194380309);
73      }

```

```
74     if (Print_results == 1)
75     {
76         printf("\n X = %7.4f mm, Y = %7.4f mm ",x,y);
77         printf("\n Azimuth = %10.8f, Elevation = %10.8f", AzEl[0],AzEl[1]);
78         printf("\n\n");
79         Wait_for_key_Press();
80     }
81     }
82
83     /* Function AzEl2AB computes disk positions from azimuth and elevation */
84     void AzEl2AB(double Azimuth, double Elevation,
85         long int *DiskA, long int *DiskB,
86         double *Delta, double *Alpha,
87         int *Number_of_Solutions)
88     {
89         double theta;
90         double Ratio;
91         int Array, Alt_element;
92
93         theta=atan2(tan(Elevation),tan(Azimuth));
94         Ratio=sqrt((pow(tan(Azimuth),2) + pow(tan(Elevation),2)))/(2.0*1.396*tan(11.0*PI/180.0));
95
96         Delta[0]=90.0+180.0/PI * (-theta + acos(Ratio));
97         Alpha[0]=90.0+360.0/PI * asin(Ratio);
98
99         Delta[1]=90.0-180.0/PI * (theta + acos(Ratio));
100        Alpha[1]=90.0-360.0/PI * asin(Ratio);
101
102        if (fabs(Delta[0]) >240.0)
103        {
104            if (Delta[0] < 0) Delta[0] = Delta[0] + 360.0;
105            else Delta[0] = Delta[0] - 360.0;
106        }
107        if (fabs(Delta[1]) >240.0)
108        {
109            if (Delta[1] < 0) Delta[1] = Delta[1] + 360.0;
110            else Delta[1] = Delta[1] - 360.0;
111        }
112
113        Alt_element = 2;
114
115        if (fabs(Delta[0]) >= 120)
116        {
117            if (Delta[0] < 0) Delta[Alt_element] = Delta[0] + 360.0;
118            else Delta[Alt_element] = Delta[0] - 360.0;
119            Alpha[Alt_element] = Alpha[0];
120            Alt_element = Alt_element + 1;
121        }
122
123        if (fabs(Delta[1]) >= 120)
124        {
125            if (Delta[1] < 0) Delta[Alt_element] = Delta[1] + 360.0;
126            else Delta[Alt_element] = Delta[1] - 360.0;
127            Alpha[Alt_element] = Alpha[1];
128            Alt_element = Alt_element + 1;
129        }
130
131        for (Array = 0; Array <= Alt_element-1; Array++)
132        {
133            DiskA[Array]=(long int) (1966.666666667*(Delta[Array]));
134            DiskB[Array]=(long int) (1966.666666667*(Delta[Array])
135                -720.5459770*((Alpha[Array])-90));
136        }
137        *Number_of_Solutions = Alt_element - 1;
138        return;
139    }
140
```

```

1
2  /* discrete.C: controls all functions of the dc servo motors */
3
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <graph.h>
9  #include <dos.h>
10 #include <time.h>
11
12 /* Mask_bit array is used to mask off bits that are not needed for test
13    these will be for switches 1-18 */
14 int Mask_bit[19] = {0x00,0x00,0x00,
15                    0x40,0x80,0x20,
16                    0x10,0x00,0x00,
17                    0x04,0x08,0x02,
18                    0x01,0x00,0x00,
19                    0x40,0x80,0x20,
20                    0x10};
21
22 int Activation_bit[19] = {0x00,0x00,0x00,
23                         0x40,0x80,0x00,
24                         0x00,0x00,0x00,
25                         0x04,0x08,0x00,
26                         0x00,0x00,0x00,
27                         0x40,0x80,0x00,
28                         0x00};
29
30
31 int Index_Mask_bit[7] = {0x0000,0x0004,0x0008,0x2000,
32                        0x1000,0x8000,0x4000};
33
34 int Detect_Bit_5V = {0x01};
35 int Detect_Bit_12V = {0x02};
36
37 int Sync_Pulse_Detect[3] = {0x0600,0x0500,0x0300};
38 int Sync_pulse_data[16] = {0,0,0,3,0,2,1,0,0,0,0,0,0,0,0,0};
39 int Sync_Number;
40
41 extern unsigned Port_A_Address;      /* Address of port A on 8255 */
42 extern unsigned Port_B_Address;      /* Address of port B on 8255 */
43 extern unsigned Port_C_Address;      /* Address of port C on 8255 */
44 extern unsigned Port_373;            /* Address for the input post 373*/
45
46 /* Declare the following variables external these var. are declared in motor.c */
47 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
48 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
49 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
50
51 extern Set_to_actual;
52
53 unsigned Port_Address; /* Variable used to store address of
54                        port to be read or written to */
55 int elapsedtime = 0;
56
57
58 /* Function used to test for switch activation */
59 int Check_for_switch_Activation(int Switch_Number)
60 {
61     int Detect_count = 0;
62     int Switch_data = 0;
63     int Switch_detect = 0;
64     int read_loop;
65
66     if ((Switch_Number >= 0) && (Switch_Number <= 14))
67         Port_Address = Port_A_Address;
68     else
69         Port_Address = Port_B_Address;
70
71     Switch_data = inp(Port_Address) & Mask_bit[Switch_Number];
72
73     Detect_count = 0;

```

```
74
75     if (Switch_data == Activation_bit[Switch_Number])
76     {
77         /* Delay reading the switch position for x amount of time */
78         for (elapsedtime = 20000; elapsedtime > 0; elapsedtime = elapsedtime - 1);
79
80         for (read_loop = 1; read_loop <= 3; read_loop++)
81         {
82             /* Delay reading the switch position for x amount of time */
83             for (elapsedtime = 5000; elapsedtime > 0; elapsedtime = elapsedtime - 1);
84             Switch_data = inp(Port_Address) & Mask_bit[Switch_Number];
85
86             if (Switch_data == Activation_bit[Switch_Number])
87             {
88                 Detect_count = Detect_count+1;
89             }
90         } /* End of for (read_loop = 1; read_loop <= 3; read_loop++) */
91
92         /* If switch was closed for 3 reads indicate that switch closure
93            was detected */
94
95         if (Detect_count == 3)
96             Switch_detect = 1;
97         else Switch_detect = 0;
98     } /* End of (Switch_data == Activation_bit[Switch_Number]) */
99
100     return Switch_detect;
101 } /* End of function Check_for_switch_Activation */
102
103
104     /* Run_Switch_Bit will be used to test the switches to see which ones
105        are activated */
106
107 int Run_Switch_Bit(int Channel_to_test,int Display_output)
108 {
109     int Number_of_Switch[12] =(3,4,5,6,9,10,11,12,15,16,17,18);
110     int Act_switch = 0;
111     int Switch_flag = 0;
112     int Loop_count = 0;
113     int Start,End;
114     float Channel_number;
115
116     if (Channel_to_test == 1)
117     {
118         Start = 0; End = 3; Channel_number = 2.5;
119     }
120     else if (Channel_to_test == 2)
121     {
122         Start = 4; End = 7; Channel_number = 1.0;
123     }
124     else
125     {
126         Start = 8; End = 11; Channel_number = 4.0;
127     }
128
129     for (Loop_count = Start; Loop_count <= End; Loop_count++)
130     {
131         Act_switch = Check_for_switch_Activation(Number_of_Switch[Loop_count]);
132         if (Act_switch == 1)
133         {
134             printf("\nSwitch %i activated \n", Number_of_Switch[Loop_count]);
135             Switch_flag = 1;
136         }
137     } /* End of for loop_count */
138
139     if ((Switch_flag == 0) && (Display_output == 1))
140     printf("\n No Switches found to be activated on Channel %2.1f\n",
141           Channel_number);
142
143     return Switch_flag;
144
145 } /* End of function Run_Switch_Bit */
146
147 /* Check_for_Index_Mark checks for the occurs of an index mark */
```



```
148
149
150 int Check_for_Index_Mark(int Motor_Number)
151 {
152
153     int Index_data = 0;
154     int Index_detect = 0;
155
156     if ((Motor_Number == 1) || (Motor_Number == 2))
157         Port_Address = Port_B_Address;
158     else
159         Port_Address = Port_373;
160
161     Index_data = inpw(Port_Address) & Index_Mask_bit[Motor_Number];
162
163     if (Index_data == 0)
164     {
165         /* Delay reading the switch position for x amount of time */
166         for (elapsedtime = 1000; elapsedtime > 0; elapsedtime = elapsedtime - 1);
167
168         Index_data = inpw(Port_Address) & Index_Mask_bit[Motor_Number];
169         if (Index_data == 0)
170             Index_detect = 1;
171     }
172
173     return Index_detect;
174
175 } /* End of function Check_for_index mark */
176
177 /* Function checks for to see if 5v supply is on */
178 int Check_for_5Volts(void)
179 {
180
181     int Volt_Data = 0;
182     int Active_5Volts = 0;
183
184     Port_Address = Port_B_Address;
185
186     Volt_Data = inpw(Port_Address) & Detect_Bit_5V;
187
188     if (Volt_Data == Detect_Bit_5V)
189         Active_5Volts = 1;
190     return Active_5Volts;
191 }
192
193 /* Function checks for to see if +24v supply is on */
194 int Check_for_24Volts(void)
195 {
196
197     int Volt_Data = 0;
198     int Active_24Volts = 0;
199
200     Port_Address = Port_B_Address;
201
202     Volt_Data = inpw(Port_Address) & Detect_Bit_12V;
203
204     if (Volt_Data == Detect_Bit_12V)
205         Active_24Volts = 1;
206     return Active_24Volts;
207 }
208
209 int Check_for_Motor_Power(void)
210 {
211     int option_key;
212     int loop;
213     int exit= 0;
214     char key_buffer[80];
215     int Pwr_On24,Pwr_On5 = 0;
216     Pwr_On24 = Check_for_24Volts();
217     Pwr_On5 = Check_for_5Volts();
218
219     if ((Pwr_On24 != 1) || (Pwr_On5 != 1))
220     {
221         /* Set all position registers to actual position before turning power */
```

```
222 Set_Motor_Position(Motor_Pair1_Address,Set_to_actual);
223 Set_Motor_Position(Motor_Pair2_Address,Set_to_actual);
224 Set_Motor_Position(Motor_Pair3_Address,Set_to_actual);
225
226 while ((Pwr_On24 != 1) && (Pwr_On5 != 1))
227 {
228     /* Clear screen and ask for what channel we need to change */
229     _clearscreen( _GCLEARSCREEN ); /* Clear screen for next output */
230     _settextposition(1,1);
231
232     printf("\n No Power to motor detected\n");
233     printf("\n1. Turn on digital and motor power and press '1' to continue \n");
234     printf("\n2. Exit to main menu \n");
235
236     printf("\nSelect the one of the above options\n");
237     option_key = -1; /* Reset value key value */
238
239     do
240     {
241         gets( key_buffer);
242         option_key = atoi(key_buffer);
243     } while ((option_key <= 0) || (option_key > 2));
244
245     if (option_key == 2)
246     {
247         exit = 1; break;
248     }
249
250     Pwr_On24 = Check_for_24Volts();
251     Pwr_On5 = Check_for_5Volts();
252
253     } /* end of while (Power_On != 1)*/
254 }
255 return exit;
256 }
257
258 /* Function Check_for_Active_Sync_Pulse checks for active sync pulse */
259 int Check_for_Active_Sync_Pulse(void)
260 {
261
262     Sync_Number = (inpw(Port_373) & 0x0F00) >> 8;
263
264     return Sync_pulse_data[Sync_Number];
265
266 }
267
```

```
1
2  /* Display routine is used to provide the menu for manual control of the motors */
3
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <graph.h>
9  #include <dos.h>
10 #include <time.h>
11 #include <stdlib.h>
12
13 /* Declare the following variables external these var. are declared in motor.c */
14 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
15 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
16 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
17 extern unsigned Control_word_address; /* Address of Control word on 8255 */
18 extern unsigned Port_A_Address; /* Address of port A on 8255 */
19 extern unsigned Port_B_Address; /* Address of port B on 8255 */
20 extern unsigned Port_C_Address; /* Address of port C on 8255 */
21 extern unsigned Port_373; /* Address of 74LS373 input port */
22 extern int Set_to_actual;
23 extern int Reset_to_zero;
24 extern long int Starting_Position[2][3];
25 extern long int Current_Position[2][3];
26 extern double Azimuth[3],Elevation[3];
27 double Current_Azimuth,Current_Elevation;
28
29 int Stop_Serial_Data[3] = {0xFFBF,0xFF7F,0xFFEF};
30 int Initiate_Serial_Tran[3] = {0x40,0x80,0x10};
31
32 long int timelapsed = 2000000; /* Variable for delay function */
33 int Channel_number; /* Number of channel to be selected */
34 int Error = 0;
35 int Passed_check; /* Variable used for receiving error number */
36 extern int C_Port_Data;
37 int Step,Menu_option,Initial_alignment,Loop,Solution;
38 long int DiskAstep,DiskBstep,Power_on;
39 double El_Az[2];
40 double Alpha_Delta[2];
41 double Temp_Azimuth[3];
42 double Temp_Elevation[3];
43 double Stop_point;
44
45 /* Array of data used to read in RS232 data back from the DSP */
46 extern struct Channel_data
47 {
48     unsigned Time_tag;
49     double V_error;
50     double U_error;
51     double Signal_strength;
52     signed char Transmit_Data;
53 } Channel[3],Test[3];
54
55 void Wait_for_key_Press(void);
56 void Process_Main_Option (int Key_pressed);
57 int Display_Main_Menu(void);
58
59 char key_buffer[80];
60 extern FILE *Manual_test_ptr;
61 extern FILE *AzEl_Ptr;
62
63
64 int Display_Main_Menu(void)
65 {
66     int key;
67
68     /*Print the main menu to the screen */
69
70
71     _clearscreen( _GCLEARSCREEN );
72     _settextposition(1,1); /*Position the cursor at position 1,1)*/
73
```

```

74
75     printf( "          **          MANUAL CONTROL MENU          **\n\n");
76
77     printf( "1. Set mode of motor controllers\n");
78     printf( "2. Read Status registor\n");
79     printf( "3. Write to analog port\n");
80     printf( "4. Write to PWM port\n");
81     printf( "5. Set sampler timer\n");
82     printf( "6. Set digital filter parameters\n");
83     printf( "7. Read digital filter parameters\n");
84     printf( "8. Set motor acceleration\n");
85     printf( "9. Set motor velocity\n");
86     printf( "10. Read motor acceleration\n");
87     printf( "11. Read max motor velocity\n");
88     printf( "12. Read actual position of motor\n");
89     printf( "13. Clear position registor \n");
90     printf( "14. Write final position to controller \n");
91     printf( "15. Read final position to controller \n");
92     printf( "16. Write command position to controller \n");
93     printf( "17. Read command position to controller \n");
94     printf( "18. Run motor in trapezoidal mode \n");
95     printf( "19. Read input discretes \n");
96     printf( "20. Align channel\n");
97     _settextposition(3,42);
98     printf( "21. Move optics arm");
99     _settextposition(4,42);
100    printf( "22. Exercise channel through limits");
101    _settextposition(5,42);
102    printf( "23. Test safety switches");
103    _settextposition(6,42);
104    printf( "24. Test serial link ");
105    _settextposition(7,42);
106    printf( "25. Test PC to PC serial link ");
107    _settextposition(8,42);
108    printf( "26. Backlash test ");
109    _settextposition(9,42);
110    printf( "27. Step response test ");
111    _settextposition(10,42);
112    printf( "28. Eccentricity test ");
113    _settextposition(11,42);
114    printf( "29. Compute Azimuth & Elevation ");
115    _settextposition(12,42);
116    printf( "30. Change Starting Azimuth & Elevation");
117    _settextposition(13,42);
118    printf( "31. Exercise multiple channels");
119    _settextposition(14,42);
120    printf( "32. Reset motor controllers");
121    _settextposition(15,42);
122    printf( "33. Check power supply discretes");
123    _settextposition(16,42);
124    printf( "34. Move arm to designated Az and El");
125    _settextposition(17,42);
126    printf( "35. Select channels to run");
127
128    _settextposition(23,1);
129    printf("Select the one of the above options\n");
130    key = 0; /* Reset value key value */
131
132    /* Read in data from the keyboard that is an interger format */
133
134    gets( key_buffer);
135    key = atoi(key_buffer);
136
137    if (key_buffer[0] == 'e') Close_down();
138
139    /* Test to see if it is within range of menu values */
140    while ((key <= 0) || (key > 35))
141    {
142        gets( key_buffer);
143        key = atoi(key_buffer);
144        if (key_buffer[0] == 'e') Close_down();
145    }
146    return key;
147 }

```

```
148 /* Output the main menu to the screen and make a selection */
149
150 int Display_Secondary_Menu(void)
151 {
152     int key;
153     int Command_options[15] = {0,21,34,22,31,29,20,23,24,25,33,26,27,28,32};
154     /*Print the main menu to the screen */
155
156
157     _clearscreen( _GCLEARSCREEN );
158     _settextposition(1,1); /*Position the cursor at position 1,1)*/
159
160     printf( "          **          MANUAL CONTROL MENU          **\n\n");
161
162     printf( " * Move Channel Commands *\n");
163
164     printf( "1. Move optics arm delta & alpha increment\n");
165     printf( "2. Move arm to designated Az and El\n");
166     printf( "3. Exercise single channel through limits\n");
167     printf( "4. Exercise all channels through limits\n");
168     printf( "5. Compute channel Az & El\n\n");
169
170     printf( " * Alignment Command *\n");
171     printf( "6. Channel alignment\n\n");
172
173     printf( " * System Test Commands *\n");
174     printf( "7. Motor safety switch test\n");
175     printf( "8. DSP to PC serial link test\n");
176     printf( "9. Host PC to Monitoring PC serial link test\n");
177     printf( "10. Power monitoring discrete test\n");
178     printf( "11. Motor drive backlash test \n");
179     printf( "12. Motor step response test \n");
180     printf( "13. Channel eccentricity test \n");
181     printf( "14. Reset Motor Command");
182
183
184     _settextposition(23,1);
185     printf("Select the one of the above options\n");
186     key = 0; /* Reset value key value */
187
188
189     /* Read in data from the keyboard that is an interger format */
190
191     gets( key_buffer);
192     key = atoi(key_buffer);
193
194     if (key_buffer[0] == 'e') Close_down();
195
196     /* Test to see if it is within range of menu values */
197     while ((key <= 0) || (key > 15))
198     {
199         gets( key_buffer);
200         key = atoi(key_buffer);
201         if (key_buffer[0] == 'e') Close_down();
202     }
203     /* The following command is used to convert secondary menu options
204        to reflect options from the primary menu */
205
206     key = Command_options[key];
207     return key;
208 }
209
210
211 void Process_Main_Option (int Key_pressed)
212 {
213     int read_data[3];
214     long int read_long_data[3];
215     int write_data[3];
216     long int write_long_data[3];
217     int Address_of_Motor;
218     int time,Switch_activated,print_results;
219     char buffer[80];
220     int Channels_to_Exercise[4];
221     int Channel;
```

```
222
223
224 /* Clear screen and ask for what channel we need to change */
225 _clearscreen( _GCLEARSCREEN ); /* Clear screen for next output */
226
227 Write_system_data(); /*Save position of motors to file */
228
229 if ((Key_pressed == 19) || (Key_pressed == 30) || (Key_pressed == 35)
230     || (Key_pressed == 31) || (Key_pressed == 32) || (Key_pressed == 33))
231 {
232     _settextposition(10,1); /*Position the cursor at position 1,1)*/
233     Channel_number = 1;
234 }
235 else
236 {
237     _settextposition(10,1); /*Position the cursor at position 1,1)*/
238     printf("\nEnter the number of the channel (1, 2, 3) ");
239
240     gets(buffer);
241     Channel = atoi(buffer);
242
243     Channel_number = 0;
244
245     /* Set address of motor based on the input from the keyboard */
246     if (Channel == 2)
247     {
248         Address_of_Motor = Motor_Pair1_Address;
249         Channel_number = 1;
250     }
251     else if (Channel == 1)
252     {
253         Address_of_Motor = Motor_Pair2_Address;
254         Channel_number = 2;
255     }
256     else if (Channel == 3)
257     {
258         Address_of_Motor = Motor_Pair3_Address;
259         Channel_number = 3;
260     }
261 } /* End of Key_pressed != 19 */
262
263
264 /* Check to see that +24 volt power is on */
265 /* If exit was chosen jump to main menu */
266
267 if ((Key_pressed == 20) || (Key_pressed == 21) || (Key_pressed == 22)
268     || (Key_pressed == 26) || (Key_pressed == 27) || (Key_pressed == 28)
269     || (Key_pressed == 29) || (Key_pressed == 31) || (Key_pressed == 34))
270 {
271     Passed_check = 0;
272     Passed_check = Check_for_Motor_Power();
273     if (Passed_check == 1) Key_pressed = 100;
274 }
275
276 if ((Channel_number >= 1) & (Channel_number <= 3))
277 {
278     switch (Key_pressed)
279     {
280         case 1: /* Set control mode of motor controllers */
281         {
282             int option_key;
283             unsigned statusword;
284
285             printf("\n ** Select mode of operation ** \n");
286
287             printf("\n1. Software reset of controller \n");
288             printf("\n2. Initialization/Idle mode \n");
289             printf("\n3. Align Mode \n");
290             printf("\n4. Control Mode \n");
291
292             printf("Select the one of the above options\n");
293
294         }
295     }
```

```
296     option_key = 0; /* Reset value key value */
297
298     /* Read in data from the keyboard that is an integer format */
299     gets( key_buffer);
300     option_key = atoi(key_buffer);
301
302     /* Test to see if it is within range of menu values */
303     while ((option_key <= 0) || (option_key > 4))
304
305     gets( key_buffer);
306     option_key = atoi(key_buffer);
307
308     statusword = Read_status_reg(Address_of_Motor);
309
310     if (option_key == 1)
311         Write_to_program_counter(Address_of_Motor,0x0000);
312
313     else if (option_key == 2)
314         Write_to_program_counter(Address_of_Motor,0x0101);
315
316     else if (option_key == 3) {
317         if ((statusword & 0x2020) == 0x2020) /* Check for Idle mode */
318             Write_to_program_counter(Address_of_Motor,0x0202);
319         else {
320             printf("\nController is not in IDLE mode. Set controller to IDLE mode to continue \n");
321             Wait_for_key_Press(); }
322     }
323
324     else if (option_key == 4) {
325         if ((statusword & 0x2020) == 0x2020) /* Check for Idle mode */
326         {
327             int zero[3] = {0,0,0};
328             int zero_long[3] = {0,0,0};
329
330             /* Clear flag to disable trapezoidal move */
331             Write_to_flag_registor (Address_of_Motor,0x0000);
332
333             /* Set the controller motor control to bipolar operation */
334             Write_to_flag_registor (Address_of_Motor,0x0202);
335
336             /* Clear flag to disable proportional velocity mode */
337             Write_to_flag_registor (Address_of_Motor,0x0303);
338
339             /* Clear flag to disable intergal velocity mode */
340             Write_to_flag_registor (Address_of_Motor,0x0505);
341
342             /* Initialize command position to 0 */
343             Write_command_pos(Address_of_Motor,zero_long);
344
345             /*Initialize actual position to 0 */
346             Reset_Actual_Position(Address_of_Motor,zero);
347
348             /* Initialize final position to 0 */
349             Write_Final_pos(Address_of_Motor,zero_long);
350
351             Write_to_program_counter(Address_of_Motor,0x0303);
352         }
353     }
354     else {
355         printf("\nController is not in IDLE mode. Set controller to IDLE mode to continue \n");
356         Wait_for_key_Press(); }
357 }
358
359 break;
360 }
361
362     case 2: /* Read Status registor */
363
364     {
365         unsigned status;
366
367         status = Read_status_reg(Address_of_Motor);
368
369         printf("\n\nValue of the status registor for DiskB motor controller = %x \n",status & 0xff);
```

```
370     printf("\nValue of the status register for DiskA motor controller = %x \n",status >> 8);
371
372     Wait_for_key_Press();
373
374     break;
375 } /* End of case 2 */
376
377     case 3: /* Write to analog port */
378
379     {
380     printf("\nEnter the value to be outputted to the DiskB motor controller");
381     printf("\nanalog port Hex (0-FF)  ");
382     cscanf( "%x", &write_data[0]);
383
384     printf("\nEnter the value to be outputted to the DiskA motor controller");
385     printf("\nanalog port in Hex (0-FF)  ");
386     cscanf( "%x", &write_data[1]);
387
388     Write_to_8bit_port(Address_of_Motor,(write_data[0] & 0x00ff)|(write_data[1] << 8));
389
390     break;
391 } /* End of case 3 */
392
393
394 case 4: /* Write to PWM port */
395
396 {
397     printf("\nEnter the value to be outputted to the DiskB motor controller");
398     printf("\npwm port Hex (0-FF)  ");
399     cscanf( "%x", &write_data[0]);
400
401     printf("\nEnter the value to be outputted to the DiskA motor controller");
402     printf("\npwm port in Hex (0-FF)  ");
403     cscanf( "%x", &write_data[1]);
404
405     Write_to_PWM_port(Address_of_Motor,(write_data[0] & 0x00ff)|(write_data[1] << 8));
406
407     break;
408
409 } /* End of case 4 */
410
411
412     case 5: /* Set sampler timer */
413
414     {
415
416     printf("\nEnter the value to be outputted to the timer of the controller in hex\n");
417     cscanf( "%x", &write_data[0]);
418
419     Write_to_sampler_timer(Address_of_Motor,write_data[0]);
420
421     break;
422
423 } /* End of case 5 */
424
425
426     case 6: /* Set digital filter parameters */
427
428     {
429
430     printf("\nSame filter parameters will be used for both");
431     printf("\nEnter the pole for the digital filter in hex\n");
432     cscanf( "%x", &write_data[0]);
433
434     printf("\nEnter the zero for the digital filter in hex\n");
435     cscanf( "%x", &write_data[1]);
436
437     printf("\nEnter the gain for the digital filter in hex\n");
438     cscanf( "%x", &write_data[2]);
439
440     Write_Filter_Pole(Address_of_Motor,write_data[0]);
441     Write_Filter_Zero(Address_of_Motor,write_data[1]);
442     Write_Filter_Gain(Address_of_Motor,write_data[2]);
443
```



```
444     break; /* Exit the case statement at this point */
445
446 } /* End of case 6 */
447
448
449     case 7: /* Read digital filter parameters */
450
451 {
452     /* Read filter data pack from controller*/
453     read_data[0] = Read_Filter_Zero(Address_of_Motor);
454     read_data[1] = Read_Filter_Pole(Address_of_Motor);
455     read_data[2] = Read_Filter_Gain(Address_of_Motor);
456
457     printf("\n\n ** DiskB Motor Controller Filter Parameters **\n");
458     printf(" Zero of digital filter (Hex) = %x \n", (read_data[0] & 0xff));
459     printf(" Pole of digital filter (Hex) = %x \n", (read_data[1] & 0xff));
460     printf(" Gain of digital filter (Hex) = %x \n", (read_data[2] & 0xff));
461
462     printf("\n\n ** DiskA Motor Controller Filter Parameters **\n");
463     printf(" Zero of digital filter (Hex) = %x \n", ((read_data[0] & 0xff00) >> 8));
464     printf(" Pole of digital filter (Hex) = %x \n", ((read_data[1] & 0xff00) >> 8));
465     printf(" Gain of digital filter (Hex) = %x \n", ((read_data[2] & 0xff00) >> 8));
466
467     Wait_for_key_Press(); /* Wait for a key to be pressed before continuing */
468
469     break;
470
471 }
472
473     case 8: /* Set motor acceleration */
474
475 {
476     printf("\nEnter the acceleration for DiskB motor in hex \n");
477     cscanf( "%x", &write_data[0]);
478
479     printf("\nEnter the acceleration for DiskA motor in hex \n");
480     cscanf( "%x", &write_data[1]);
481
482     /* Output the values of acceleration to the controllers */
483     Write_Max_Accel(Address_of_Motor, write_data);
484
485     break;
486 } /* End of case 8 */
487
488
489     case 9: /* Set motor velocity */
490
491 {
492
493     printf("\nEnter the max velocity for DiskB motor (0-7FH) \n");
494     cscanf( "%x", &write_data[0]);
495
496     printf("\nEnter the max velocity for DiskA motor (0-7FH) \n");
497     cscanf( "%x", &write_data[1]);
498
499     /* Output the velocity value to the controllers*/
500     Write_Max_Velocity(Address_of_Motor, (write_data[0] & 0x007f) |
501                                     (write_data[1] << 8) & 0x7FFF);
502
503     break;
504 } /* End of case 9 */
505
506     case 10: /* Read max motor acceleration*/
507
508 {
509
510
511     Read_Max_Accel(Address_of_Motor, read_data);
512     printf("\n DiskB motor acceleration is set at (Hex) %x \n", read_data[0]);
513     printf("\n DiskA motor acceleration is set at (Hex) %x \n", read_data[1]);
514     Wait_for_key_Press();
515
516     break;
517 } /* End of case 10 */
```

```
518
519     case 11: /* Read PWM port */
520
521 {
522     read_data[0] = Read_Max_Velocity(Address_of_Motor);
523
524     printf("\n DiskB motor max. velocity is set at (Hex) %x",read_data[0] & 0xff);
525     printf("\n DiskA motor max. velocity is set at (Hex) %x \n\n",read_data[0] >> 8);
526     Wait_for_key_Press();
527
528     break;
529 } /* End of case 11 */
530
531     case 12: /*Read actual position of motor */
532
533 {
534     getch(); /*Clear all data out of buffer */
535
536     timelaped = 20000;
537
538     while( !kbhit()) /* Wait for key on keyboard to be pressed */
539     {
540         Read_Actual_Pos(Address_of_Motor,read_long_data);
541
542         printf("\n DiskB motor position = %li ",read_long_data[0]);
543         printf(" DiskA motor position = %li ",read_long_data[1]);
544
545         /* delay the program be requesting new data */
546         for (timelaped = 20000; timelaped > 0; timelaped = timelaped - 1)
547             time = time +1;
548         time = 0;
549     }
550     /* Use getch to throw key away. */
551     getch();
552
553     break;
554 } /* End of case 12 */
555
556 case 13:
557
558 {
559     Reset_Actual_Position(Address_of_Motor,0x0000);
560     break;
561 } /* End of case 13 */
562
563     case 14: /* Write final position */
564
565 {
566     /* Enter position to which motor will be moved to */
567
568     printf("\nEnter new position for DiskB motor movement - \n");
569     gets( key_buffer);
570     write_long_data[0] = atol(key_buffer);
571
572     /* Enter position to which DiskA motor will be moved to */
573     printf("\nEnter new position for DiskA motor movement - \n");
574     gets( key_buffer);
575     write_long_data[1] = atol(key_buffer);
576
577     /* Output the final position tho the controller*/
578     Write_Final_pos(Address_of_Motor,write_long_data);
579
580     break;
581 } /* End of case 14 */
582
583     case 15: /* Read final position register */
584
585 {
586     Read_Final_Pos(Address_of_Motor,read_long_data);
587
588     printf("\n The final position of the DiskB motor is = %li ",read_long_data[0]);
589     printf("\n The final position of the DiskA motor is = %li \n",read_long_data[1]);
590
591     Wait_for_key_Press();
```

```
592
593     break;
594 } /* End of case 15 */
595
596     case 16: /* Write command position */
597
598     {
599         break;
600
601         printf("\nEnter the command position to run the DiskB motor to \n");
602         gets( key_buffer);
603         write_long_data[0] = atol(key_buffer);
604
605         printf("\nEnter the command position to run the DiskA motor to \n");
606         gets( key_buffer);
607         write_long_data[0] = atol(key_buffer);
608
609         /* Output the command position from the controller*/
610         Write_command_pos(Address_of_Motor,write_long_data);
611
612         break;
613 } /* End of case 16 */
614
615     case 17: /* Read command position register */
616
617     {
618         Read_Command_Pos(Address_of_Motor,read_long_data);
619
620         printf("\n The command position of the DiskB motor is = %li ",read_long_data[0]);
621         printf("\n The command position of the DiskA motor is = %li \n",read_long_data[1]);
622         Wait_for_key_Press();
623
624         break;
625 } /* End of case 17 */
626
627     case 18: /* Start motor */
628     /* Start the motor for a trapzoidel move */
629     {
630         Write_to_flag_registor (Address_of_Motor,0x0808);
631         break;
632     } /* End of case 18 */
633
634     case 19: /* Read out discretes and print to the screen */
635
636     {
637         getch(); /*Clear all data out of buffer */
638
639         timelaspd = 100000;
640
641         while( !kbhit()) /* Wait for key on keyboard to be pressed */
642         {
643             read_data[0] = inp(Port_A_Address);
644             read_data[1] = inp(Port_B_Address);
645             read_data[2] = inpw(Port_373);
646             printf("\n Port A Discretes = %x ",read_data[0] & 0xff);
647             printf("    Port B Discretes = %x \t Controller Port = %x",
648                 read_data[1],read_data[2]);
649
650             /* delay the program be requesting new data */
651             for (timelaspd = 25; timelaspd > 0; timelaspd = timelaspd - 1)
652                 time = time +1;
653             time = 0;
654         }
655         /* Use getch to throw key away. */
656         getch();
657         break;
658     } /* End of case 19 */
659
660     case 20: /* Align selected channel */
661
662     {
663         printf("\n ** Alignment options ** \n");
664         printf("\n 1.) Encoder initialization ");
665         printf("\n 2.) Initial alignment calibration ");
666         printf("\n 3.) Exit without alignment ");
```

```
666     printf("\n Select desired option ");
667
668     do {
669         gets( key_buffer);
670         Menu_option = atoi(key_buffer);
671     } while ((Menu_option <= 0) || (Menu_option > 3));
672
673     if (Menu_option == 2)
674     {
675         Initial_alignment = 1;
676         Set_Motor_Position(Address_of_Motor,Reset_to_zero);
677         Align_Channel (Address_of_Motor,Channel_number,Initial_alignment);
678     }
679     else if (Menu_option == 1)
680     {
681         Initial_alignment = 0;
682         Align_Channel (Address_of_Motor,Channel_number,Initial_alignment);
683     }
684
685     break;
686 }
687
688 case 21: /* Move optics arm */
689 {
690     long int Offset_Position[3];
691     long int alpha = 0;
692     long int delta = 0;
693     int in_range_flag = 0; /* Flag indicating data inputted is in range */
694
695     while (in_range_flag == 0)
696     {
697         printf("\nEnter change in delta position of arm \n");
698
699         gets( key_buffer);
700         delta = atol(key_buffer);
701
702         printf("\nEnter change in alpha position of arm \n");
703
704         gets( key_buffer);
705         alpha = atol(key_buffer);
706
707         if ((delta <= 480) && (delta >= -480) &&
708             (alpha <= 40) && (alpha >= -40))
709             in_range_flag = 1;
710         else printf("\nInvalid range select again!! \n");
711     } /* End of while in_range_flag == 0 */
712
713     /* Read in actual position of the DiskA and DiskB motors for offset */
714     Read_Actual_Pos(Address_of_Motor,Offset_Position);
715
716     /* Compute position for DiskB motor */
717     write_long_data[0] = (1967 * -delta) + (alpha * 769) + Offset_Position[0];
718
719     /* Compute position for DiskA motor command */
720     write_long_data[1] = Offset_Position[1] + (1967 * delta);
721
722     /* Output the final position tho the controller*/
723     Write_Final_pos(Address_of_Motor,write_long_data);
724
725     /* Start up motors for arm movement */
726     Write_to_flag_registor (Address_of_Motor,0x0808);
727     break;
728 } /* End of case 21 */
729
730 case 22: /* Exercise channel through its limits */
731 {
732     Exercise_Channel(Address_of_Motor,Channel_number);
733     break;
734 }
735
736 case 23: /* Run test on switches for designated channel */
```

```
740 {
741     printf("\n    Switch Test Results    \n");
742     print_results = 1;
743     Switch_activated = Run_Switch_Bit(Channel_number, print_results);
744     Wait_for_key_Press();
745     break;
746 }
747
748     case 24: /* Run test on serial link */
749 {
750     Init_Com_port(); /*Initialize serial port */
751     ComFlushRx(); /* Flush out data before starting loop */
752     printf("\nPower receiver electronics\n");
753     Wait_for_key_Press();
754     printf("\n\nTesting channel %i's serial link. Please wait!\n",
755           Channel);
756     Error = Test_Serial_Link(Channel_number, C_Port_Data);
757     if (Error == 0) printf("\n\nSerial link communications good\n");
758     ComCloseAll();
759     Wait_for_key_Press();
760     break;
761 }
762
763     case 25: /* Run test on serial link */
764 {
765     Init_Com_port(); /*Initialize serial port */
766     ComFlushRx(); /* Flush out data before starting loop */
767     printf("\n\nHit any key to exit option");
768     while( !kbhit()) /* Wait for key on keyboard to be pressed */
769     {
770         Error = Test_Serial_Link(Channel_number, C_Port_Data);
771     }
772     getch();
773     ComCloseAll();
774     break;
775 }
776
777     case 26: /* Run backlash test */
778 {
779     printf("\n 1.) Test Backlash Disk B channel (U_error)");
780     printf("\n 2.) Test Backlash Disk A channel");
781     printf("\n 3.) Test Backlash Disk A & B channel together (V_error)");
782     printf("\n 4.) Test Backlash Disk A & B channel in opposite directions");
783     printf("\n 5.) Exit test");
784     printf("\n Select desired option  ");
785     do {
786         gets( key_buffer);
787         Menu_option = atoi(key_buffer);
788     } while ((Menu_option <= 0) || (Menu_option > 5));
789     if (Menu_option != 5)
790     {
791         Init_Com_port(); /*Initialize serial port */
792         ComFlushRx(); /* Flush out data before starting loop */
793         Run_Back_Lash_Test(Address_of_Motor, C_Port_Data,
794                           Channel_number, Menu_option);
795         ComCloseAll();
796     }
797     break;
798 }
799
800     case 27: /* Test motor for step response */
801 {
```

```
814     printf("\n      Command move test options ");
815     printf("\n 1.) Test Disk B motor");
816     printf("\n 2.) Test Disk A motor");
817     printf("\n 3.) Test both Disk B & A motors running in same direction");
818     printf("\n 4.) Test both Disk B & A motors running in opposite direction");
819     printf("\n      Trapezoidal move test options ");
820     printf("\n 5.) Test Disk B motor");
821     printf("\n 6.) Test Disk A motor");
822     printf("\n 7.) Test both Disk B & A motors running in same direction");
823     printf("\n 8.) Test both Disk B & A motors running in opposite direction");
824
825     printf("\n 9.) Exit test");
826     printf("\n Select desired option  ");
827
828     do {
829         gets( key_buffer);
830         Menu_option = atoi(key_buffer);
831     } while ((Menu_option <= 0) || (Menu_option > 9));
832
833     if (Menu_option != 9)
834     {
835         printf("\n Enter step response value Range 1 - 1000 counts ");
836
837         do {
838             gets( key_buffer);
839             Step = atoi(key_buffer);
840         } while ((Step <= 0) || (Step > 1000));
841
842         fprintf(Manual_test_ptr,"\n Step response value = %i",Step);
843         Run_Step_Test(Address_of_Motor,Channel_number,Step,Menu_option);
844
845     } /* End of if Menu_option != 4 */
846     break;
847 }
848
849     case 28: /* Run eccentricity test on gears */
850     {
851
852         Init_Com_port(); /*Initialize serial port */
853         ComFlushRx(); /* Flush out data before starting loop */
854
855         Run_Eccen_Test(Address_of_Motor,C_Port_Data,
856                        Channel_number);
857
858         ComCloseAll();
859         break;
860     }
861
862     case 29:
863     {
864
865         El_Az[0] = 0.0; /* Set azimuth and elevation array values to zero */
866         El_Az[1] = 0.0;
867         DiskAstep = 0;
868         DiskBstep = 0;
869
870         Read_AzEl_data();
871
872
873         /* Read data from system file on present position */
874         Read_Current_System_data();
875
876
877         /* Store values read in program into temporary array */
878         for (loop = 0; loop <= 2; loop++)
879         {
880             Temp_Azimuth[loop] = Azimuth[loop];
881             Temp_Elevation[loop] = Elevation[loop];
882         }
883
884
885         printf("\n\n      *** Compute Azimuth and Elevation parameters *** \n\n");
886         printf("\n 1.) Use current disks position on Channel %i",Channel);
887         printf("\n 2.) Enter disks positions by hand for Channel %i",Channel);
```

```
888     printf("\n 3.) Compute AZ & EL for all three channels using current positions");
889     printf("\n 4.) Exit option");
890     printf("\n\n Select desired options\n");
891
892     do
893     {
894         gets( key_buffer);
895         Menu_option = atoi(key_buffer);
896     }
897     while ((Menu_option <= 0) || (Menu_option > 4));
898
899     /* Exit case statement at this time */
900     if (Menu_option == 4) break;
901
902     if (Menu_option == 2)
903     {
904         printf("\nEnter IstepA position for computing Az & El \n");
905         gets( key_buffer);
906         DiskAstep = atol(key_buffer);
907
908         printf("\nEnter IstepB position for computing Az & El \n");
909         gets( key_buffer);
910         DiskBstep = atol(key_buffer);
911
912         printf("\n\n Azimuth & Elevation for Channel %i",Channel);
913
914         Compute_AZ_EL (DiskAstep,DiskBstep,1,El_Az,Alpha_Delta);
915         Temp_Azimuth[Channel_number -1] = El_Az[0]*57.29577951;
916         Temp_Elevation[Channel_number -1] = El_Az[1]*57.29577951;
917     }
918
919     if (Menu_option == 1)
920     {
921
922         DiskAstep = Current_Position[1][Channel_number - 1];
923         DiskBstep = Current_Position[0][Channel_number - 1];
924
925         printf("\n\n Azimuth & Elevation for Channel %i",Channel);
926
927         Compute_AZ_EL (DiskAstep,DiskBstep,1,El_Az,Alpha_Delta);
928         Temp_Azimuth[Channel_number -1] = El_Az[0]*57.29577951;
929         Temp_Elevation[Channel_number -1] = El_Az[1]*57.29577951;
930
931     }
932
933
934     if (Menu_option == 3)
935     {
936         for (loop = 0; loop <= 2; loop++)
937         {
938             if (loop == 0) Channel = 2;
939             else if (loop == 1) Channel = 1;
940             else if (loop == 2) Channel = 3;
941
942             DiskAstep = Current_Position[1][loop];
943             DiskBstep = Current_Position[0][loop];
944
945             printf("\n\n Azimuth & Elevation for Channel %i",Channel);
946
947             Compute_AZ_EL (DiskAstep,DiskBstep,1,El_Az,Alpha_Delta);
948             Temp_Azimuth[loop] = El_Az[0]*57.29577951;
949             Temp_Elevation[loop] = El_Az[1]*57.29577951;
950
951         }
952     }
953
954     Menu_option = 0;
955
956     printf("\n1.) Save changes to file ");
957     printf("\n2.) Continue without saving changes");
958     printf("\n Select option\n");
959     do
960     {
961         gets( key_buffer);
```

```
962     Menu_option = atoi(key_buffer);
963 }
964 while ((Menu_option <= 0) || (Menu_option > 2));
965
966 if (Menu_option == 1)
967 {
968     for (loop = 0; loop <= 2; loop++)
969     {
970         Azimuth[loop] = Temp_Azimuth[loop];
971         Elevation[loop] = Temp_Elevation[loop];
972     }
973     Write_AzEl_data();
974 }
975
976 break;
977 }
978
979     case 30:
980     {
981         Select_Azel_Changes(); /*Request user for new azimuth and elevation parameters*/
982         break;
983     }
984
985     case 31:
986     {
987         Channels_to_Exercise[0] = 0; /* Disable channel */
988         Channels_to_Exercise[1] = 1;
989         Channels_to_Exercise[2] = 1;
990         Channels_to_Exercise[3] = 1; /* Enable channel */
991
992         Exercise_Multiple_Channels(Channels_to_Exercise);
993         break;
994     }
995
996     case 32: /* Reset all motor controllers to present actual position */
997     {
998         Address_of_Motor = Motor_Pair1_Address;
999         Set_Motor_Position(Address_of_Motor, Set_to_actual);
1000         Address_of_Motor = Motor_Pair2_Address;
1001         Set_Motor_Position(Address_of_Motor, Set_to_actual);
1002         Address_of_Motor = Motor_Pair3_Address;
1003         Set_Motor_Position(Address_of_Motor, Set_to_actual);
1004         break;
1005     }
1006
1007     case 33: /* Check power supply */
1008     {
1009         Power_on = Check_for_24Volts();
1010         if (Power_on == 1)
1011             printf("\n +24 Volt power supply on");
1012         else printf("\n +24 Volt power supply off");
1013
1014         Power_on = Check_for_5Volts();
1015         if (Power_on == 1)
1016             printf("\n +/-5 Volt power supply on\n\n");
1017         else printf("\n +/-5 Volt power supply off\n\n");
1018         Wait_for_key_Press();
1019
1020         break;
1021     }
1022
1023 case 34: /* Move arm to new position */
1024 {
1025     Menu_option = 0;
1026
1027     Read_AzEl_data();
1028
1029     /* Read data from system file on present position */
1030     Read_Current_System_data();
1031
1032     /* Use getch to throw key away. */
1033     while (kbhit()) getch();
1034
1035     printf("\n1.) Enter absolute AZ & EL values ");
```



```
1036     printf("\n2.) Move arm with keyboard arrows");
1037     printf("\n3.) Run scan pattern");
1038     printf("\n4.) Exit without moving arm");
1039
1040     printf("\n Select option\n");
1041
1042     do
1043     {
1044         gets( key_buffer);
1045         Menu_option = atoi(key_buffer);
1046     }
1047     while ((Menu_option <= 0) || (Menu_option > 4));
1048
1049     /* Use getch to throw key away. */
1050     while (kbhit()) getch();
1051
1052
1053     if (Menu_option == 1)
1054     {
1055         Move_to_New_AZEL(Channel_number,Address_of_Motor);
1056     }
1057
1058     if ((Menu_option == 2) || (Menu_option == 3))
1059     {
1060
1061         DiskAstep = Current_Position[1][Channel_number-1];
1062         DiskBstep = Current_Position[0][Channel_number-1];
1063
1064         Compute_AZ_EL (DiskAstep,DiskBstep,0,El_Az,Alpha_Delta);
1065         Current_Azimuth = El_Az[0];
1066         Current_Elevation = El_Az[1];
1067
1068         Init_Com_port(); /*italize serial port */
1069
1070         if (Menu_option == 2)
1071             Position(Current_Azimuth, Current_Elevation,
1072                     Channel_number,Address_of_Motor);
1073         else
1074         {
1075             printf("\nEnter level of trip point for scan (+0.2 to +39.0 Volts)\n");
1076
1077             Stop_point = 0;
1078
1079             while ((Stop_point < 0.2) || (Stop_point > 39.0))
1080             {
1081                 gets(key_buffer);
1082                 Stop_point = atof(key_buffer);
1083             }
1084
1085             /* Use getch to throw key away. */
1086             while (kbhit()) getch();
1087
1088             Run_Scan_Pattern (Current_Azimuth,Current_Elevation,Channel_number,
1089                             Address_of_Motor, Stop_point);
1090         }
1091
1092         while (kbhit()) getch();
1093
1094     }
1095     printf("\n1.) Save AZ & EL values for tracking ");
1096     printf("\n2.) Exit without saving Az & EL");
1097
1098     printf("\n Select option\n");
1099
1100     do
1101     {
1102         gets( key_buffer);
1103         Menu_option = atoi(key_buffer);
1104     }
1105     while ((Menu_option <= 0) || (Menu_option > 2));
1106
1107     if (Menu_option == 1)
1108     {
1109         Write_system_data();
```

```
1110     Read_Current_System_data();
1111
1112     DiskAstep = Current_Position[1][Channel_number-1];
1113     DiskBstep = Current_Position[0][Channel_number-1];
1114
1115     Compute_AZ_EL (DiskAstep,DiskBstep,0,El_Az,Alpha_Delta);
1116     Azimuth[Channel_number-1] = El_Az[0]*57.29577951;
1117     Elevation[Channel_number-1] =El_Az[1]*57.29577951;
1118
1119     Write_AzEl_data();
1120     Write_AzEl_Track_Data();
1121
1122 }
1123 ComCloseAll();
1124 }
1125 break;
1126 }
1127
1128     case 35:
1129     {
1130     Select_Channels_to_Run();
1131     }
1132
1133     default:
1134     break;
1135
1136     } /* End of switch statement */
1137
1138     } /* End of if Channel_number >=1) & (Channel_number <= 3 */
1139
1140 } /* End of function Call_Routine */
1141
1142
1143 void Wait_for_key_Press(void)
1144 {
1145     {
1146     printf("Press any key to continue");
1147     /* Display message until key is pressed. */
1148
1149     while( !kbhit()); /* Wait for key on keyboard to be pressed */
1150
1151     /* Use getch to throw key away. */
1152     while (kbhit()) getch();
1153     }
1154 }
1155
1156
1157
1158
1159
1160
1161
```

```
1
2  /* File.C: Used to open all nessacary files for diagonoastic */
3
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <dos.h>
9  #include <time.h>
10 #include <cport.h>
11 #include <math.h>
12 #include <stdlib.h>
13 #include <graph.h>
14
15 FILE *output_ptr; /* Set up pointer for file to output error data */
16 FILE *Data_Ptr; /* Set up pointer for file to output test data */
17 FILE *System_Ptr; /* Set up pointer for file that contains system data */
18 FILE *AzEl_Ptr; /* Set up pointer for file that contains Azimuth &
19 Elevation data for all three channels*/
20 FILE *Track_AzEl_Ptr; /* Set up pointer for tracking par. for tracking software */
21 FILE *Align_Ptr; /* Set up pointer with alignment info */
22 FILE *Manual_test_ptr; /* Set up pointer for file to output error data */
23 FILE *Input_Ptr; /* Set up pointer for data that contains input data
24 to */
25
26 long int Disk_Position[2];
27 long int Disk_Leo_Position[2];
28 long int Starting_Position[2][3];
29 long int Current_Position[2][3];
30 long int Leo_Position[2][3];
31 long int Align_Position[2][3];
32 double Azimuth[3],Elevation[3];
33 double Track_Az[3],Track_El[3];
34 int Chan,Chan_Address;
35 int Channel_data;
36
37 /* Declare the following variables external these var. are declared in motor.c */
38 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
39 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
40 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
41 extern long int Offset_Position[3][4];
42 extern int Selected_Channels[3][2];
43
44 void Read_system_data(void);
45 void Open_files(int track);
46 void Write_system_data(void);
47
48
49 void Open_files(int track)
50 {
51     /* If tracking mode is used open these additional files */
52
53     if (track == 1)
54     {
55         if ((output_ptr=fopen("c:\\nasa\\output.txt","at"))==NULL)
56         {
57             printf("Cannot open file\n");
58             exit(1);
59         }
60
61         if ((Data_Ptr=fopen("c:\\nasa\\Test.dat","at"))==NULL)
62         {
63             printf("Cannot open file\n");
64             exit(1);
65         }
66     }
67
68     /* Open only if using manual mode */
69
70     if (track == 0)
71     {
72         if ((Manual_test_ptr=fopen("c:\\nasa\\manual.dat","a+t"))==NULL)
73         {
```

```
74     printf("Cannot open file\n");
75     exit(1);
76 }
77 }
78
79     if ((System_Ptr=fopen("c:\\nasa\\System.dat","r+t"))==NULL)
80     {
81     printf("Cannot open file\n");
82     exit(1);
83     }
84
85     if ((AzEl_Ptr=fopen("c:\\nasa\\SysAzEl.dat","r+t"))==NULL)
86     {
87     printf("Cannot open file\n");
88     exit(1);
89     }
90
91     if ((Track_AzEl_Ptr=fopen("c:\\nasa\\AzEl.dat","r+t"))==NULL)
92     {
93     printf("Cannot open file\n");
94     exit(1);
95     }
96
97
98     if ((Align_Ptr=fopen("c:\\nasa\\Align.dat","r+t"))==NULL)
99     {
100    printf("Cannot open file\n");
101    exit(1);
102    }
103
104    if ((Input_Ptr=fopen("c:\\nasa\\Input.dat","r+t"))==NULL)
105    {
106    printf("Cannot open file\n");
107    exit(1);
108    }
109
110    Read_system_data();
111
112    /* Print header file for data */
113    fprintf(Data_Ptr,"\nTimeTag Sig Lev V_Error U_Error");
114    fprintf(Data_Ptr,"      ActB      DiskBC      IstepB      ActA      DiskAC      IStepA");
115    }
116
117 void Read_system_data(void)
118 {
119
120    rewind (System_Ptr);
121    for (Chan = 0; Chan <= 2; Chan++)
122    {
123        fscanf(System_Ptr," %ld %ld\n",&Disk_Position[1],&Disk_Position[0]);
124        Starting_Position[0][Chan] = Disk_Position[0];
125        Starting_Position[1][Chan] = Disk_Position[1];
126    }
127    for (Chan = 0; Chan <= 2; Chan++)
128    {
129        fscanf(System_Ptr,"%i\n",&Channel_data);
130        Selected_Channels[Chan][1] = Channel_data;
131    }
132 }
133
134 void Read_Current_System_data(void)
135 {
136    rewind (System_Ptr);
137    for (Chan = 0; Chan <= 2; Chan++)
138    {
139        fscanf(System_Ptr," %ld %ld\n",&Disk_Position[1],&Disk_Position[0]);
140        Current_Position[0][Chan] = Disk_Position[0];
141        Current_Position[1][Chan] = Disk_Position[1];
142    }
143 }
144
145 void Write_system_data(void)
146 {
```

```
148      /* Move file point to begining of file */
149      rewind (System_Ptr);
150
151      for (Chan = 1; Chan <= 3; Chan++)
152      {
153          switch (Chan)
154          {
155              case 1:
156              {
157                  Chan_Address = Motor_Pair1_Address; break;
158              }
159              case 2:
160              {
161                  Chan_Address = Motor_Pair2_Address; break;
162              }
163              case 3:
164              {
165                  Chan_Address = Motor_Pair3_Address; break;
166              }
167          }
168
169          Read_Actual_Pos(Chan_Address,Disk_Position);
170
171          Disk_Position[0] = Starting_Position[0][Chan-1] - Disk_Position[0];
172          Disk_Position[1] = Starting_Position[1][Chan-1] + Disk_Position[1];
173
174          fprintf(System_Ptr,"%ld %ld\n",Disk_Position[1],Disk_Position[0]);
175      }
176
177      /* Write out data of active channels */
178      for (Chan = 0; Chan <= 2; Chan++)
179          fprintf(System_Ptr," %i\n", Selected_Channels[Chan][1]);
180
181  } /* End of function Write_system data */
182
183  /* Reads in data for user changing of azimuth and elevation errors*/
184  void Read_AzEl_data(void)
185  {
186      /* Move file point to begining of file */
187      rewind (AzEl_Ptr);
188
189      for (Chan = 0; Chan <= 2; Chan++)
190      {
191          fscanf(AzEl_Ptr," %lf\n",&Azimuth[Chan]);
192          fscanf(AzEl_Ptr," %lf\n",&Elevation[Chan]);
193      }
194  } /* End of Read_AzEl_data function*/
195
196
197
198  /* Writes out new azimuth and elevation data */
199  void Write_AzEl_data(void)
200  {
201      /* Move file point to begining of file */
202      rewind (AzEl_Ptr);
203
204      for (Chan = 0; Chan <= 2; Chan++)
205      {
206          fprintf(AzEl_Ptr," %lf\n",Azimuth[Chan]);
207          fprintf(AzEl_Ptr," %lf\n",Elevation[Chan]);
208      }
209  } /* End of Write_AzEl_data function */
210
211
212
213  /* Writes out new azimuth and elevation data */
214  void Write_AzEl_Track_Data(void)
215  {
216      int loop = 0;
217      double dummy = 0;
218      int zero;
219      int switch_channel[3] = {1,0,2};
220
221      /* Move file point to begining of file */
```

```
222     rewind (Track_AzEl_Ptr);
223
224     for (Chan = 0; Chan <= 2; Chan++)
225     {
226         if (Selected_Channels[switch_channel[Chan]][1] != 0)
227         {
228             fprintf(Track_AzEl_Ptr," %lf\n",Azimuth[switch_channel[Chan]]);
229             fprintf(Track_AzEl_Ptr," %lf\n",Elevation[switch_channel[Chan]]);
230             loop++;
231         }
232     }
233     if (loop < 3)
234     {
235         for (zero = 1; zero <= (3 - loop); zero++)
236         {
237             fprintf(Track_AzEl_Ptr," %lf\n",dummy);
238             fprintf(Track_AzEl_Ptr," %lf\n",dummy);
239         }
240     }
241 } /* End of Write_AzEl_data function */
242
243 /* Writes out new Alignment Data to file*/
244 void Write_Align_data(void)
245 {
246     /* Move file point to begining of file */
247     rewind (Align_Ptr);
248
249     for (Chan = 0; Chan <= 2; Chan++)
250     {
251         fprintf(Align_Ptr," %ld\n",Align_Position[0][Chan]);
252         fprintf(Align_Ptr," %ld\n",Align_Position[1][Chan]);
253     }
254 }
255
256 /* Reads in alignment data for all channels */
257 void Read_Align_data(void)
258 {
259     /* Move file point to begining of file */
260     rewind (Align_Ptr);
261
262     for (Chan = 0; Chan <= 2; Chan++)
263     {
264         fscanf(Align_Ptr," %ld\n",&Align_Position[0][Chan]);
265         fscanf(Align_Ptr," %ld\n",&Align_Position[1][Chan]);
266     }
267 }
268
269 void Change_Input_Data(void)
270 {
271     /* Create and open temporary file. */
272     FILE *outfile;
273     char tmp[120];
274     int loop, line_count,Channels_To_Run;
275
276     outfile = tmpfile();
277     /* Get each line from input and write to output. */
278     for (line_count = 1; line_count <= 12; line_count++)
279     {
280         fgets( tmp, 120, Input_Ptr);
281         fputs( tmp, outfile );
282     }
283
284     Channels_To_Run = 0;
285
286     for (loop =0; loop <= 2; loop++)
287     {
288         if (Selected_Channels[loop][1] != 0)
289             Channels_To_Run = Channels_To_Run +1;
290     }
291 }
```

```
296         rewind (Input_Ptr);
297         rewind (outfile);
298
299         for (line_count = 1; line_count <= 12; line_count++)
300         {
301             fgets( tmp, 120, outfile);
302             /* Add 48 to the number of channels to indicate the ascii char.*/
303             if (line_count == 12) tmp[0] = Channels_To_Run + 48;
304             fputs( tmp, Input_Ptr );
305         }
306     }
```

```

1
2  /* Display routine is used to provide the menu for manual control of the motors */
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <graph.h>
9  #include <dos.h>
10 #include <time.h>
11 #include <math.h>
12 #define BLK SZ 8
13 #define Pi 3.141592654
14
15 /* Declare the following variables external these var. are declared in motor.c */
16 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
17 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
18 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
19
20 extern unsigned Port_A_Address; /* Address of port A on 8255 */
21 extern unsigned Port_B_Address; /* Address of port B on 8255 */
22 extern unsigned Port_C_Address; /* Address of port C on 8255 */
23
24 long int timelapsed;
25 int delay;
26
27 int Reset_to_zero = 1; /* These 2 variables will be used global for setting motor position */
28 int Set_to_actual = 0;
29
30 struct rccoord pos;
31
32 void Initialize_Controllers (void);
33
34 void Initialize_Controllers (void )
35 {
36     int Ch_num,Address_of_Channel;
37     int long zero[2] = {0,0};
38     int Accel_data[2];
39
40     /* Declare the digital filter parameters */
41     /* Format of the Filter pole array are as follows
42     Filter_Pole[0] Bits 16 - 9 Channel 1 Arm motor pole
43     Filter_Pole[0] Bits 8 - 1 Channel 1 Drum motor pole
44     Filter_Pole[1] Bits 16 - 9 Channel 2 Arm motor pole
45     Filter_Pole[1] Bits 8 - 1 Channel 2 Drum motor pole
46     Filter_Pole[2] Bits 16 - 9 Channel 3 Arm motor pole
47     Filter_Pole[2] Bits 8 - 1 Channel 3 Drum motor pole */
48
49     int Filter_Pole[3] = {0xe6e6,0xe6e6,0xe6e6};
50
51     /* Format of the Filter zero array are as follows
52     Filter_Zero[0] Bits 16 - 9 Channel 1 Arm motor Zero
53     Filter_Zero[0] Bits 8 - 1 Channel 1 Drum motor Zero
54     Filter_Zero[1] Bits 16 - 9 Channel 2 Arm motor Zero
55     Filter_Zero[1] Bits 8 - 1 Channel 2 Drum motor Zero
56     Filter_Zero[2] Bits 16 - 9 Channel 3 Arm motor Zero
57     Filter_Zero[2] Bits 8 - 1 Channel 3 Drum motor Zero */
58
59     int Filter_Zero[3] = {0xd3d3,0xd3d3,0xd3d3};
60
61     /* Format of the Filter Gain array are as follows
62     Filter_Gain[0] Bits 16 - 9 Channel 1 Arm motor Gain
63     Filter_Gain[0] Bits 8 - 1 Channel 1 Drum motor Gain
64     Filter_Gain[1] Bits 16 - 9 Channel 2 Arm motor Gain
65     Filter_Gain[1] Bits 8 - 1 Channel 2 Drum motor Gain
66     Filter_Gain[2] Bits 16 - 9 Channel 3 Arm motor Gain
67     Filter_Gain[2] Bits 8 - 1 Channel 3 Drum motor Gain */
68
69     int Filter_Gain[3] = {0x4040,0x9090,0x9090};
70
71     /* Format of the Sampler_Timer array are as follows
72     Sample_Time[0] Bits 16 - 9 Channel 1 Arm motor timer
73     Sample_Time[0] Bits 8 - 1 Channel 1 Drum motor timer

```



```
74 Sample_Time[1] Bits 16 - 9 Channel 2 Arm motor timer
75 Sample_Time[1] Bits 8 - 1 Channel 2 Drum motor timer
76 Sample_Time[2] Bits 16 - 9 Channel 3 Arm motor timer
77 Sample_Time[2] Bits 8 - 1 Channel 3 Drum motor timer */
78
79     int Sample_Time[3] = {0x0F0F,0x0F0F,0x0F0F};
80
81
82     /* Format of the Max_velocity array are as follows
83 Max_Velocity[0] Bits 16 - 9 Channel 1 Arm motor max velocity
84 Max_Velocity[0] Bits 8 - 1 Channel 1 Drum motor max velocity
85 Max_Velocity[1] Bits 16 - 9 Channel 2 Arm motor max velocity
86 Max_Velocity[1] Bits 8 - 1 Channel 2 Drum motor max velocity
87 Max_Velocity[2] Bits 16 - 9 Channel 3 Arm motor max velocity
88 Max_Velocity[2] Bits 8 - 1 Channel 3 Drum motor max velocity */
89
90     int Max_Velocity[3] = {0x0202,0x0202,0x0303};
91
92     /* Format of the Max_Accel array are as follows
93
94 Max_Accel[0] Bits 16 - 9 Channel 1 Arm motor max Accel
95 Max_Accel[1] Bits 8 - 1 Channel 1 Drum motor max Accel
96 Max_Accel[2] Bits 16 - 9 Channel 2 Arm motor max Accel
97 Max_Accel[3] Bits 8 - 1 Channel 2 Drum motor max velocity
98 Max_Accel[4] Bits 16 - 9 Channel 3 Arm motor max Accel
99 Max_Accel[5] Bits 8 - 1 Channel 3 Drum motor max Accel */
100
101     int Max_Accel[6] = {0x1000,0x1000,0x1000,0x1000,0x1000,0x1000};
102
103     for (Ch_num = 0; Ch_num < 3; Ch_num = Ch_num +1)
104     {
105         /* Set address of motor based on loop counter */
106
107         if (Ch_num == 0)
108             Address_of_Channel = Motor_Pair1_Address;
109         else if (Ch_num == 1)
110             Address_of_Channel = Motor_Pair2_Address;
111         else if (Ch_num == 2)
112             Address_of_Channel = Motor_Pair3_Address;
113
114         /* Do a soft reset on the drum and motor controller of channel x */
115         Write_to_program_counter(Address_of_Channel,0x0000);
116
117         /* Clear flag to disable trapezoidal move */
118         Write_to_flag_register (Address_of_Channel,0x0000);
119
120         /* Set the controller motor control to bipolar operation */
121         Write_to_flag_register (Address_of_Channel,0x0202);
122
123         /* Clear flag to disable proportional velocity mode */
124         Write_to_flag_register (Address_of_Channel,0x0303);
125
126         /* Clear flag to disable intergal velocity mode */
127         Write_to_flag_register (Address_of_Channel,0x0505);
128
129         /* Initialize command position to 0 */
130         Write_command_pos(Address_of_Channel,zero);
131
132         /*Initialize actual position to 0 */
133         Reset_Actual_Position(Address_of_Channel,zero);
134
135         /* Initialize final position to 0 */
136         Write_Final_pos(Address_of_Channel,zero);
137
138         /* Set up sample timer for the controller */
139         Write_to_sampler_timer(Address_of_Channel,Sample_Time[Ch_num]);
140
141
142         /* Initialize the filter parameters for the digital filter */
143         Write_Filter_Pole(Address_of_Channel,Filter_Pole[Ch_num]);
144         Write_Filter_Zero(Address_of_Channel,Filter_Zero[Ch_num]);
145         Write_Filter_Gain(Address_of_Channel,Filter_Gain[Ch_num]);
146
147         Accel_data[0] = Max_Accel[Ch_num];
```

```

148     Accel_data[1] = Max_Accel[Ch_num + 1];
149
150     /* Output the values of acceleration to the controllers */
151     Write_Max_Accel(Address_of_Channel,Accel_data);
152
153     /* Output the velocity value to the controllers*/
154     Write_Max_Velocity(Address_of_Channel,Max_Velocity[Ch_num]);
155
156     /* Put the motor controller into the control mode */
157     Write_to_program_counter(Address_of_Channel,0x0303);
158
159     } /* End of for loop Ch_num < 3 */
160
161 } /* End of function Call_Routine */
162
163
164 int Set_Motor_Position(unsigned Address_of_Reset,int Reset)
165 {
166     long int Position[2] = {0,0};
167
168     /* If reset pin is equal to 1 reset all position counters to zero */
169     if (Reset == 1)
170         Reset_Actual_Position(Address_of_Reset,Position);
171     else
172         /* Read actual position of motors */
173         Read_Actual_Pos(Address_of_Reset,Position);
174
175     /* Put motor controller pair into idle mode */
176     Write_to_program_counter(Address_of_Reset,0x0101);
177
178     for (timelaspded = 5000; timelaspded > 0; timelaspded = timelaspded - 1);
179
180     /* Initialize command position to actual position */
181     Write_command_pos(Address_of_Reset,Position);
182
183     /* Initialize final position to actual position */
184     Write_Final_pos(Address_of_Reset,Position);
185
186     /* Put motor controller pair back into control mode */
187     Write_to_program_counter(Address_of_Reset,0x0303);
188
189     for (timelaspded = 10000; timelaspded > 0; timelaspded = timelaspded - 1);
190
191 }
192
193
194
195 /* Function Reached_Commanded_Pos checks to see if the motors have reached
196 their commanded position */
197 int Reached_Commanded_Pos(unsigned Mot_Add,long int *Target_value,int Channel,int Difference)
198 {
199     int Target_Reached = 1;
200     long int Pres_Pos[2];
201     long int Delta_pos1,Delta_pos0;
202     int delay;
203     int Check_count = 0;
204
205     while ((Target_Reached == 1) && (Check_count < 6))
206     {
207         Read_Actual_Pos(Mot_Add,Pres_Pos);
208
209         /* Compute deltas between present and target position */
210         Delta_pos1 = labs(Target_value[1] - Pres_Pos[1]);
211         Delta_pos0 = labs(Target_value[0] - Pres_Pos[0]);
212
213         if ((Delta_pos1 > Difference) || (Delta_pos0 > Difference))
214             ( Target_Reached = 0; Check_count == 0; )
215         else
216             ( Target_Reached = 1; Check_count ++;
217               for (delay = 0; delay < 5000; delay++);
218             )
219     }
220     return Target_Reached;
221 }

```

222
223
224
225

```
1
2  /* Manual.C: Main executive for manual control of the dc servo motors */
3
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <graph.h>
9  #include <dos.h>
10 #include <time.h>
11
12 extern FILE *Manual_test_ptr; /* Set up pointer for file to output error data */
13 extern FILE *Align_Ptr;
14 extern FILE *System_Ptr;
15 extern FILE *AzEl_Ptr;
16 extern FILE *Track_AzEl_Ptr;
17
18 extern int C_Port_Data = 0;
19 extern unsigned Port_C_Address;
20 extern Set_to_actual;
21
22 /* Declare the following variables external these var. are declared in motor.c */
23 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
24 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
25 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
26
27
28 int Pwr_24Volts_On = 0x20; /*Variable used to turn on 24 volt supply */
29 int Pwr_24Volts_Off = 0xDF; /*Variable used to turn off 24 volt supply */
30 long int Offset_Position[3][4];
31 int mode = {0};
32
33 extern unsigned Control_word_address; /* Address to which 8255 command word
34                                     is being set */
35 extern unsigned Control_word; /* Sets all ports A,B to inputs & C to outputs */
36
37 extern struct Channel_data
38 {
39     unsigned Time_tag;
40     double V_error;
41     double U_error;
42     double Signal_strength;
43     signed char Transmit_Data;
44 } Channel[3],Test[3];
45
46
47 void main ()
48 {
49     int Key_data= 0;
50     int Primary_Menu = 0;
51
52     /* Assign channel assignments to data array */
53     Test[0].Transmit_Data=0;
54     Test[1].Transmit_Data=1;
55     Test[2].Transmit_Data=2;
56
57     _setvideomode(_HERCMONO); /* Set mode of graphics to Hercules monitor */
58
59     /* Open all files as required mode = 0 Indicates manual mode */
60     Open_files(mode);
61
62     /* Output control word to 8255 */
63     outp(Control_word_address, Control_word);
64
65     /* Initialize motor controllers */
66     Initialize_Controller();
67
68     /* Turn on 24 Volt power supply */
69     C_Port_Data = C_Port_Data | Pwr_24Volts_On;
70     outp(Port_C_Address,C_Port_Data);
71
72     while(1)
73     {
```

```
74     if (Primary_Menu == 1)
75         Key_data = Display_Main_Menu();
76     else
77         Key_data = Display_Secondary_Menu();
78
79     Process_Main_Option(Key_data); /*Process key that was pressed */
80 }
81
82 }
83
84
85 void Close_down(void)
86 { int ch;
87
88     ComCloseAll();          /* Close all serial ports */
89
90     /* Turn off 24 Volt power supply */
91     C_Port_Data = C_Port_Data & Pwr_24Volts_Off;
92     outp(Port_C_Address,C_Port_Data);
93
94     /* Set all position registers to actual position to stop motors */
95     Set_Motor_Position(Motor_Pair1_Address,Set_to_actual);
96     Set_Motor_Position(Motor_Pair2_Address,Set_to_actual);
97     Set_Motor_Position(Motor_Pair3_Address,Set_to_actual);
98
99     Write_system_data();      /* Read position of motors and store data */
100    fclose (Manual_test_ptr); /* Close file when done */
101    fclose (Align_Ptr);
102    fclose (System_Ptr);
103    fclose (AzEl_Ptr);
104    fclose (Track_AzEl_Ptr);
105
106    exit (1);
107
108 }
109
110
111
```

```
1
2 #include <stdio.h>
3 #include <conio.h>
4 #include <ctype.h>
5 #include <time.h>
6 #include <graph.h>
7 #include <dos.h>
8 #include <time.h>
9 #include <float.h>
10 #include <stdlib.h>
11
12 extern long int Starting_Position[2][3];
13 extern double Azimuth[3],Elevation[3];
14
15
16 /* Selected_Channel array is defined as follows first element indicates
17 electronic & software channel and second element denotes optical channel */
18
19 int Selected_Channels[3][2] = {{1,2},{2,1},{3,3}};
20 int Channel_to_Run[4] = {0,0,0,0};
21 int Number_of_Channels_To_Run = 3;
22
23 char key_buffer[80];
24 char buffer[80];
25
26 int exit_loop,loop_test;
27 int option_key,chan;
28 unsigned statusword;
29 double temp_az[3];
30 double temp_el[3];
31
32 void Select_Azel_Changes (void)
33 {
34     exit_loop = 0;
35
36     Read_AzEl_data();
37
38     /* Store values read in program into temporary array */
39     for (chan = 0; chan <= 2; chan++)
40     {
41         temp_az[chan] = Azimuth[chan];
42         temp_el[chan] = Elevation[chan];
43     }
44
45     while (exit_loop != 1)
46     {
47         _clearscreen( _GCLEARSCREEN );
48         _settextposition(1,1); /*Position the cursor at position 1,1)*/
49
50         printf("\n          ** Azimuth and Elevation options menu ** \n\n");
51
52         printf("Channel 1 Azimuth = %8.6f,(Deg.) Channel 1 Elevation = %8.6f,(Deg.)\n"
53             ,temp_az[1],temp_el[1]);
54         printf("Channel 2 Azimuth = %8.6f,(Deg.) Channel 2 Elevation = %8.6f,(Deg.)\n"
55             ,temp_az[0],temp_el[0]);
56         printf("Channel 3 Azimuth = %8.6f,(Deg.) Channel 3 Elevation = %8.6f,(Deg.)\n"
57             ,temp_az[2],temp_el[2]);
58
59         printf("\n1.  Change Azimuth and Elevation for Channel 1 \n");
60         printf("2.  Change Azimuth and Elevation for Channel 2 \n");
61         printf("3.  Change Azimuth and Elevation for Channel 3 \n");
62         printf("4.  Continue without saving changes \n");
63         printf("5.  Save changes and continue\n");
64
65         printf("\nSelect the one of the above options\n");
66         option_key = -1; /* Reset value key value */
67
68         do
69         {
70             gets( key_buffer);
71             option_key = atoi(key_buffer);
72         } while ((option_key <= 0) || (option_key > 5));
73     }
```

```
74     switch (option_key)
75     {
76         case 1:
77             { option_key = 2; break;}
78         case 2:
79             option_key = 1;
80     }
81
82     /* Read in data from the keyboard that is an integer format */
83
84     if ((option_key == 1) || (option_key == 2) || (option_key == 3))
85     {
86         printf("\nEnter new value for azimuth in Degrees (Range +/-11)\n");
87         gets(key_buffer);
88         temp_az[option_key-1] = atof(key_buffer);
89         printf("\nEnter new value for elevation in Degrees (Range +/-11)\n");
90         gets(key_buffer);
91         temp_el[option_key-1] = atof(key_buffer);
92     }
93     else if (option_key == 4)
94     {
95         exit_loop = 1; /* Set flag to exit main loop */
96     }
97     else if (option_key == 5)
98     {
99         exit_loop = 1;
100         /* Store changes back into array*/
101         for (chan = 0; chan <= 2; chan++)
102         {
103             Azimuth[chan] = temp_az[chan];
104             Elevation[chan] = temp_el[chan];
105         }
106         Write_AzEl_data();
107         Write_AzEl_Track_Data();
108     }
109
110
111     } /* End of while exit_loop != 1*/
112 } /* End of function Select_Azel_Changes */
113
114
115 /* Function Move_to_New_AZEL used to move to new azel location */
116 void Move_to_New_AZEL(channel, Motor_Address)
117 {
118     double Delta[4], Alpha[4];
119     long int I_StepA[4], I_StepB[4];
120     long int Command_Pos[2];
121     int a,b, Number_of_Solutions, loop, Solution;
122     long int Present_Delta = 10000000;
123     double New_Azimuth = 12.0;
124     double New_Elevation = 12.0;
125
126     _clearscreen( _GCLEARSCREEN );
127     _settextposition(1,1); /*Position the cursor at position 1,1)*/
128
129     printf("\n          ** Azimuth and Elevation Channel Move ** \n\n");
130
131     printf("\nEnter new azimuth for move in Degrees \n");
132     while ((New_Azimuth > 11.0) || (New_Azimuth < -11.0))
133     {
134         gets(key_buffer);
135         New_Azimuth = atof(key_buffer);
136         if ((New_Azimuth > 11.0) || (New_Azimuth < -11.0))
137             printf("\nOut of range enter new value!! \n");
138     }
139
140     printf("\nEnter new elevation for move in Degrees\n");
141     while ((New_Elevation > 11.0) || (New_Elevation < -11.0))
142     {
143         gets(key_buffer);
144         New_Elevation = atof(key_buffer);
145         if ((New_Elevation > 11.0) || (New_Elevation < -11.0))
146             printf("\nOut of range enter new value!! \n");
147     }
```

```
148
149     New_Azimuth = New_Azimuth/57.29577951;
150     New_Elevation = New_Elevation/57.29577951;
151
152     AzEL2AB(New_Azimuth, New_Elevation, I_StepA, I_StepB, Delta,
153     Alpha,&Number_of_Solutions);
154
155     printf("\n Select one of the following alpha & delta pairs to move the channel to \n\n");
156
157     for (loop=0; loop <= Number_of_Solutions; loop++)
158     {
159         printf("%i.) Delta = %9.6f (Deg.) Alpha = %9.6f (Deg.)\n",
160         loop+1,Delta[loop],Alpha[loop]);
161     }
162     printf("%i.) Exit option without moving motors\n",Number_of_Solutions+2);
163
164     do
165     {
166         gets(key_buffer);
167         Solution = atoi(key_buffer);
168     }
169     while ((Solution < 1) || (Solution > Number_of_Solutions+2));
170
171     if (Solution != (Number_of_Solutions+2))
172     {
173         /* Set up the commanded position for the DiskB Motor */
174         Command_Pos[0] = Starting_Position[0][channel-1] - I_StepB[Solution-1];
175
176         /* Set up the commanded position for the DiskA Motor */
177         Command_Pos[1] = (-1 * Starting_Position[1][channel-1]) + I_StepA[Solution-1];
178
179         /* Output new position to the motor controller */
180         Write_Final_pos(Motor_Address,Command_Pos);
181
182         /* Start trapezoidal move */
183         Write_to_flag_register (Motor_Address,0x0808);
184     }
185 }
186
187 void Screen_Prompt(void)
188 {
189     char Key_number;
190     char tmpbuf[128];
191     _clearscreen( _GCLEARSCREEN );
192     _settextposition(10,1); /*Position the cursor at position 1,1)*/
193
194
195     printf("                Welcome to the Multiaccess terminal tracking program");
196
197     /* Display DOS-style date and time. */
198     _strtime( tmpbuf );
199     printf( "\n\nTime:\  %s\n", tmpbuf );
200     _strdate( tmpbuf );
201     printf( "Data:  %s\n", tmpbuf );
202
203
204     _settextposition(20,1); /*Position the cursor at position 1,1)*/
205     printf("Press 'e' to exit program at this time or return to continue");
206
207     while( !kbhit());
208
209     if (getch() == 'e') Close_down();
210
211 }
212
213 void Select_Channels_to_Run(void)
214 {
215     char Key_number;
216     char tmpbuf[128];
217     int Menu_option = 0;
218     int loop;
219
220     int Channel_select = 0;
```



```
222
223     Read_AzEl_data();
224
225     while (Menu_option != 4)
226     {
227         _clearscreen( _GCLEARSCREEN );
228         _settextposition(10,1); /*Position the cursor at position 1,1)*/
229
230         printf("      ** Channel selection menu **\n\n");
231         printf("The following channels have been selected: ");
232
233         if (Selected_Channels[1][1] != 0)
234             printf("%i ",Selected_Channels[1][1]);
235
236         if (Selected_Channels[0][1] != 0)
237             printf("%i ",Selected_Channels[0][1]);
238
239         if (Selected_Channels[2][1] != 0)
240             printf("%i ",Selected_Channels[2][1]);
241
242         printf("\n\n");
243
244         printf("1.) Add channel\n");
245         printf("2.) Remove channel \n");
246         printf("3.) Select all channels \n");
247         printf("4.) Continue and save selections \n");
248
249         do
250         {
251             gets( tmpbuf);
252             Menu_option = atoi(tmpbuf);
253         }
254         while ((Menu_option <= 0) || (Menu_option > 4));
255
256         if ((Menu_option == 1) || (Menu_option == 2))
257         {
258             printf("\nEnter channel number 1,2,3\n");
259
260             do
261             {
262                 gets( tmpbuf);
263                 Channel_select = atoi(tmpbuf);
264             }
265             while ((Channel_select != 1) && (Channel_select != 2)
266                   && (Channel_select != 3));
267
268             if (Menu_option == 1)
269             {
270
271                 if (Channel_select == 1) Selected_Channels[1][1] = 1;
272                 else if (Channel_select == 2) Selected_Channels[0][1] = 2;
273                 else if (Channel_select == 3) Selected_Channels[2][1] = 3;
274
275             }
276
277             if (Menu_option == 2)
278             {
279
280                 if (Channel_select == 1) Selected_Channels[1][1] = 0;
281                 else if (Channel_select == 2) Selected_Channels[0][1] = 0;
282                 else if (Channel_select == 3) Selected_Channels[2][1] = 0;
283             }
284
285             /* End of if ((Menu_option == 1) || (Menu_option == 2)) */
286
287             if (Menu_option == 3)
288             {
289                 Selected_Channels[0][1] = 2;
290                 Selected_Channels[1][1] = 1;
291                 Selected_Channels[2][1] = 3;
292             }
293
294             /* End of while menu option != 4 */
295
```

```
296      /* Compute the number of channels selected */
297
298      Number_of_Channels_To_Run = 0;
299
300      for (loop = 0; loop <= 2; loop++)
301      {
302          if (Selected_Channels[loop][1] != 0)
303      {
304          Number_of_Channels_To_Run = Number_of_Channels_To_Run + 1;
305          Channel_to_Run[Number_of_Channels_To_Run] = loop + 1;
306      }
307      }
308
309      if((Selected_Channels[0][1] != 0) && (Selected_Channels[1][1] != 0))
310      {
311          Channel_to_Run[1] = 2;
312          Channel_to_Run[2] = 1;
313      }
314
315
316      Write_system_data();
317      Write_AzEl_data();
318      Write_AzEl_Track_Data();
319      Change_Input_Data();
320
321      } /* End of function select channels */
322
323 void Check_System_Power(void)
324 {
325     int Power_on24, Power_on5;
326     Power_on5 = Check_for_5Volts();
327
328     if (Power_on5 != 1)
329     {
330         printf("\n Turn +/-5 Volt power supply on");
331         Wait_for_key_Press();
332     }
333
334     Power_on24 = Check_for_24Volts();
335
336     if (Power_on24 != 1)
337     {
338         printf("\n\n Turn +24 Volt power supply on");
339         Wait_for_key_Press();
340     }
341 }
342
343 void Check_Switch_Status()
344 {
345     int Channel_Number, Print_output, Switch_results, Switch_activated;
346     Print_output = 0;
347     Switch_results = 0;
348     Switch_activated = 0;
349
350     for (Channel_Number = 1; Channel_Number <= 3; Channel_Number++)
351     {
352         Switch_activated = Run_Switch_Bit(Channel_Number, Print_output);
353         if (Switch_activated == 1) Switch_results = 1;
354     }
355     if (Switch_results == 1)
356         Wait_for_key_Press();
357 }
358
359
360 void Check_Serial_Links(int Data_Port)
361 {
362     int Error, loop, I_channel;
363
364     {
365         for (loop = 1; loop <= Number_of_Channels_To_Run; loop ++);
366         I_channel = Channel_to_Run[loop];
367         Error = Test_Serial_Link(I_channel, Data_Port);
368     }
369     if (Error == 0) printf("\n\nSerial link communications good\n");
370     ComCloseAll();
```

370)

```
1  /*
2   The following include statements are required to allow
3   program to link up with Turbo C++ libraries */
4
5  #include <cport.h>
6  #include <bios.h>
7  #include <conio.h>
8  #include <dos.h>
9  #include <ctype.h>
10 #include <stdio.h>
11 #include <math.h>
12 #define BLK_SZ 9
13 #define Number_of_samples 24
14
15 FILE *outfile[3];
16
17 int Break_handler(void);
18 /* Start main program here at this point */
19
20 main(void)
21 {
22     /* Delcarations of these variables will cause them to be stored in CPU
23     Registers */
24
25     int rv,out,status,Channel_number;
26     int i, Channel=0;
27     char output[80];
28     signed char input_buffer[1024];
29     unsigned int rs;
30     int Elevation_error,Azimuth_error>Total_signal_strength;
31     unsigned int Time_tag[3], Old_Time_tag[3];
32     char *str = "Hello world";
33     int Frame_count[3],Process_data_flag;
34     int Relative_flag='N';
35     char Mode_flag,File_flag, log='N', a;
36     int Good_data,Length_of_Message;
37     int delay_time[10] = {0,5,4,4,3,3,2,2,1,0};
38
39     double Elevation_average,Azimuth_average,Signal_average, Signal_offset[3];
40     double Azimuth_deviation,Elevation_deviation,Signal_deviation;
41     double Azimuth_deviation_square,Elevation_deviation_square,Signal_deviation_square;
42     double Elevation,Azimuth,Signal_strength, last_signal[3];
43     double Azimuth_total[3];
44     double Azimuth_total_square[3];
45     double Elevation_total[3];
46     double Elevation_total_square[3];
47     double Signal_strength_total[3];
48     double Signal_strength_total_square[3];
49
50     /* Set variables equal to zero */
51
52     for (i=0;i<3;i++)
53     { Azimuth_total[i] = 0.0;
54       Azimuth_total_square[i] = 0.0;
55       Elevation_total[i] = 0.0;
56       Elevation_total_square[i] = 0.0;
57       Signal_strength_total[i] = 0.0;
58       Signal_strength_total_square[i] = 0.0;
59       Time_tag[i]=0;
60       Old_Time_tag[i]=0;
61     }
62     Frame_count[0] =0;
63     Frame_count[1] =0;
64     Frame_count[2] =0;
65
66     Process_data_flag = 0;
67
68     /* Set up handler for when ctrl break key is pressed */
69     /* ctrl_brk(break_handler);*/
70
71     /* Initialize RS-232 port */
72     /* rv = 2 bad 'com' parameter, rv = 3 no uart chip detected */
73     /* rv = 4 receive queue allocation error, rv = 4 transmit queue allocation error */
```

```
74  /*      the following line will be used for the compuadd computer at 115kbaud      */
75      rv = ComOpen(COM1, B19200, W8|S1|NONE, 1024, 512);
76
77  /* Set up COM1 port to be active */
78      ComActive(COM1);
79
80  /* Remove all data from the receive buffer */
81      ComFlushRx();
82
83  /* Remove all data from the transmit buffer */
84      ComFlushTx();
85
86  /* User interaction */
87      printf("Do you what to run program in continous mode? Press 'y'");
88      printf("for continous mode or any other key for manual mode");
89      Mode_flag = toupper(getch());
90
91      printf("\nDo you what to log the data to a file? Press 'y'");
92      File_flag = toupper(getch());
93
94      clrscr();
95      if (File_flag=='Y')
96      { outfile[0]=fopen("Channel0.raw","w+");
97        outfile[1]=fopen("Channel1.raw","w+");
98        outfile[2]=fopen("Channel2.raw","w+");
99        fprintf(outfile[0],"Raw El.\tRaw Az.\tRaw Sig.\tAz\tEl\tSignal\n");
100        fprintf(outfile[1],"Raw El.\tRaw Az.\tRaw Sig.\tAz\tEl\tSignal\n");
101        fprintf(outfile[2],"Raw El.\tRaw Az.\tRaw Sig.\tAz\tEl\tSignal\n");
102        log='L';
103      }
104
105      ComFlushRx();
106
107      while(1)
108      {
109
110          if (kbhit())
111          { a=getch();
112            if (a == 'e')
113            { ComCloseAll();
114              if (File_flag=='Y')
115              { close(outfile[0]);
116                close(outfile[1]);
117                close(outfile[2]);
118              }
119              exit(0);
120            }
121            if (a=='d')
122            { Process_data_flag = 1; /* If 'd' is hit process data */
123              log = 'L';
124            }
125            if ((a=='L') || (a=='l'))
126            { log='L'; }
127            if ((a=='S') || (a=='s'))
128            { log='S'; }
129            if ((a=='R') || (a=='r'))
130            { if (Relative_flag=='Y')
131              Relative_flag='N';
132              else
133              Relative_flag='Y';
134            }
135            if ((a=='Z') || (a=='z'))
136            { Signal_offset[0]=last_signal[0];
137              Signal_offset[1]=last_signal[1];
138              Signal_offset[2]=last_signal[2];
139            }
140          }
141
142          if (Relative_flag!='Y')
143          { Signal_offset[0]=0;
144            Signal_offset[1]=0;
145            Signal_offset[2]=0;
146          }
147      }
```

```
148     if (Mode_flag == 'Y')
149     { Process_data_flag = 1; }
150     else
151     { if (kbhit())
152     { a= getch();
153     if (a == 'd')
154     { Process_data_flag = 1;
155     log = 'L';
156     }
157     if ((a== 'L') || (a=='l')) log = 'L';
158
159     if ((a== 'S') || (a=='s'))
160     log = 'S';
161     }
162     } /* End of if mode_flag = Y */
163
164     Good_data = 0;
165
166     while (Good_data != 1)
167     {
168     /* Wait for data */
169     while (ComLenRx() == 0);
170
171     Length_of_Message = ComLenRx();
172
173     /* Test to see if message is longer than 9 words */
174     if (Length_of_Message > 9)
175     Length_of_Message = (Length_of_Message / 9) * 9;
176
177     /* Delay program long enough to get rest of data */
178     delay(delay_time[Length_of_Message]);
179
180     if (ComLenRx() > BLK_SZ) ComFlushRx();
181     else
182     {
183     rs = ComIn(input_buffer,BLK_SZ);
184     if ((input_buffer[0] >= 0) || (input_buffer[0] <= 2))
185     Good_data = 1;
186     else ComFlushRx();
187     }
188     }
189
190     /* Check to see if flag is set to process data */
191
192     if (Process_data_flag == 1)
193     {
194
195     /* Increment the counter for the number of frame samples
196     that have been read in */
197     Channel=(int)(input_buffer[0]) & 0x03;
198
199     Frame_count[Channel]++;
200
201     Time_tag[Channel] = (int)(input_buffer[2]) & 0xFF;
202     Time_tag[Channel] = Time_tag[Channel] << 8;
203     Time_tag[Channel] = Time_tag[Channel] | ((int)(input_buffer[1]) & 0xFF);
204
205     Elevation_error = (int)(input_buffer[4]) & 0xFF;
206     Elevation_error = Elevation_error << 8;
207     Elevation_error = Elevation_error | ((int)(input_buffer[3]) & 0xFF);
208
209     Azimuth_error = (int)(input_buffer[6]) & 0xFF;
210     Azimuth_error = Azimuth_error << 8;
211     Azimuth_error = Azimuth_error | ((int)(input_buffer[5]) & 0xFF);
212     Azimuth_error = -Azimuth_error;
213
214     Total_signal_strength = (int)(input_buffer[8]) & 0xFF;
215     Total_signal_strength = Total_signal_strength << 8;
216     Total_signal_strength = Total_signal_strength | ((int)(input_buffer[7]) & 0xFF);
217
218     /*Change integer value to a floating point value for normalizing*/
219     Signal_strength = Total_signal_strength;
220
221     if (Signal_strength <= 1e-4)
```

```
222     { Signal_strength = 0.0001; /* Set signal_strength to small
223                                     number if equal 0.0 */
224     }
225
226     /* Normalize Azimuth compute total azimuth error for n samples
227     as well as the sum of the squares */
228
229     Azimuth = Azimuth_error / Signal_strength;
230
231     Azimuth_total[Channel] += Azimuth;
232     Azimuth_total_square[Channel] += Azimuth * Azimuth;
233
234     /* Normalize Elevation compute total azimuth error for n samples
235     as well as the sum of the squares */
236
237     Elevation = Elevation_error / Signal_strength;
238
239     Elevation_total[Channel] += Elevation;
240     Elevation_total_square[Channel] += Elevation * Elevation;
241
242     /* Scale signal strength to a DC value from a 12 bit D/A with
243     a +- 10V input range */
244
245     Signal_strength = Signal_strength/204.7;
246     last_signal[Channel]=Signal_strength;
247
248     Signal_strength-=Signal_offset[Channel];
249
250
251     if (File_flag=='Y' && (log=='L' || log=='I') && Mode_flag=='Y')
252     { fprintf(outfile[Channel],"%i\t%i\t%i\t",Elevation_error,
253                                     Azimuth_error,
254                                     Total_signal_strength);
255       fprintf(outfile[Channel],"%5f\t%5f\t%5f\n",Elevation,
256                                     Azimuth,
257                                     Signal_strength);
258     }
259     /* Compute total signal strength for x samples as well as the sum
260     of the squares */
261     Signal_strength_total[Channel] += Signal_strength;
262     Signal_strength_total_square[Channel] += Signal_strength * Signal_strength;
263
264     if (Frame_count[Channel] == Number_of_samples)
265     { Azimuth_average = Azimuth_total[Channel]/Number_of_samples;
266
267       Azimuth_deviation_square = (Azimuth_total_square[Channel] /
268                                   Number_of_samples) - (Azimuth_average *
269                                                           Azimuth_average);
270       /* test to see if we have a negative number */
271       if (Azimuth_deviation_square < 0.0) Azimuth_deviation_square = 0.0;
272
273       Azimuth_deviation = sqrt(Azimuth_deviation_square);
274
275       Elevation_average = Elevation_total[Channel]/Number_of_samples;
276       Elevation_deviation_square = (Elevation_total_square[Channel] /
277                                   Number_of_samples) - (Elevation_average *
278                                                           Elevation_average);
279
280       /* test to see if we have a negative number */
281       if (Elevation_deviation_square < 0.0) Elevation_deviation_square = 0.0;
282
283       Elevation_deviation = sqrt(Elevation_deviation_square);
284
285       Signal_average = Signal_strength_total[Channel]/Number_of_samples;
286       Signal_deviation_square = (Signal_strength_total_square[Channel] /
287                                   Number_of_samples) - (Signal_average *
288                                                           Signal_average);
289
290       if (Signal_deviation_square < 0.0) Signal_deviation_square = 0.0 ;
291
292       Signal_deviation = sqrt(Signal_deviation_square);
293
294       if (Channel == 0) Channel_number = 1;
295       else if (Channel == 1) Channel_number = 0;
```

```
296         else Channel_number = 2;
297
298         gotoxy(1,(5*Channel_number) + 1);
299         clreol();
300
301         printf("Channel %i\n",Channel_number + 1);
302
303         clreol();
304
305         printf("Time = %6u v = %-1.5f  u = %-1.5f  Total = %2.3f ",
306               Time_tag[Channel],Elevation_average,Azimuth_average,
307               Signal_average);
308         if (Relative_flag=='Y')
309         { printf("R"); }
310         printf("\n");
311         clreol();
312         printf("S.Dev      v = %-1.5f  u = %-1.5f  Total = %2.4f\n",
313               Elevation_deviation,Azimuth_deviation,
314               Signal_deviation);
315         clreol();
316
317         gotoxy(1,22);
318         clreol();
319         if (File_flag=='Y' && (log=='L' || log=='l'))
320         { printf("Press 'S' to stop logging. "); }
321         else
322         { if (File_flag=='Y')
323           printf("Press 'L' to begin logging. ");
324         }
325
326         if (Relative_flag=='Y')
327         { printf("Press 'Z' to zero signal strength "); }
328         else
329         { printf("Press 'R' to toggle relative mode "); }
330
331         if (Mode_flag != 'Y')
332         { printf("\nPress 'd' to sample data");
333           fprintf(outfile[Channel],"Time = %6u v = %-1.5f  u = %-1.5f  Total = %2.3f\n",
334                 Time_tag[Channel],Elevation_average,Azimuth_average,
335                 Signal_average);
336         }
337         printf("\n");
338         /* Reset variables used to total average amd sum of squares */
339         Azimuth_total[Channel] = 0.0;
340         Azimuth_total_square[Channel] = 0.0;
341         Elevation_total[Channel] = 0.0;
342         Elevation_total_square[Channel] = 0.0;
343         Signal_strength_total[Channel] = 0.0;
344         Signal_strength_total_square[Channel] = 0.0;
345
346         /*Reset frame counter*/
347         Frame_count[Channel] = 0;
348
349         /* Reset process_data_flag */
350         Process_data_flag = 0;
351
352     } /* End of if Frame_count = Number_of_samples */
353     } /* End of if process_flag = 1 */
354 } /* End of while (1)!= */
355
356 }
357
```



```
1
2  /* Display routine is used to provide the menu for manual control of the motors */
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <graph.h>
9  #include <dos.h>
10 #include <time.h>
11 #include <math.h>
12 #define BLK_SZ 8
13 #define Pi 3.141592654
14
15 /* Declare the following variables external these var. are declared in motor.c */
16 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
17 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
18 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
19
20 extern unsigned Port_A_Address; /* Address of port A on 8255 */
21 extern unsigned Port_B_Address; /* Address of port B on 8255 */
22 extern unsigned Port_C_Address; /* Address of port C on 8255 */
23
24 extern long int Align_Position[2][3]; /* Data containtining alignment positions */
25 extern long int Starting_Position[2][3]; /* Array containing starting position */
26
27 extern long int Current_Position[2][3];
28
29 extern double El_Az[2];
30 extern double Alpha_Delta[2];
31
32 extern struct Channel_data
33 {
34     unsigned Time_tag;
35     double V_error;
36     double U_error;
37     double Signal_strength;
38     int Transmit_Data;
39 } Channel[3],Test[3];
40
41 extern FILE *Manual_test_ptr;
42 long int timelapsed;
43
44 /* The following variables are used for open loop testing */
45 double Step_A = 0.0;
46 double Step_B = 1.0;
47 long int DiskAstep,DiskBstep;
48 int Number_of_Frames = 1600;
49 int Range = 400;
50 long int Motor_Pos[2][3000];
51 long int Act_pos[2];
52 long int Zero_Position[2] = {0,0};
53 long int Step_Position[2] = {0,0};
54 int delay;
55 int Exit_requested;
56 int long Temp_actual,Temp_new;
57
58 extern int Reset_to_zero; /* These 2 variables will be used global for setting motor position */
59 extern int Set_to_actual;
60
61 struct rccoord pos;
62
63 int Sync_Pulse_Disable[3] = {0xBF,0x7F,0xEF};
64 int Sync_Pulse_Enable[3] = {0x40,0x80,0x10};
65
66 void Align_Channel(unsigned Address_of_Control,int Number_of_channel,int Stop_Flag);
67 void Compute_Open_Loop_Commands(unsigned Channel_Address,int Number_of_Motor,
68     double Alpha_arm);
69
70
71 /* Align channel will be used to align the channels to a particular point */
72 void Align_Channel(unsigned Address_of_Control,int Number_of_channel, int Store_Flag)
73
```

```
74 {
75   int long Step_size = 2000000;
76   int Index_step_size = 1;
77
78   int DiskB_Motor_number = 0;
79   int DiskA_Motor_number = 0;
80
81   int DiskA_Align_switch = 0;
82   int DiskB_Align_switch = 0;
83
84   int All_Switches_Good = 0x30; /* Bit pattern when no switches limits have been hit */
85   int Switch_activated; /* Flag used indicate switch activation */
86   int Index_Mark_Found; /* Variable used to store index data */
87   long int delta = 250;
88   long int alpha = -38;
89
90   int Delta_limit = 3; /* Set limits for delta between actual and command position */
91   long int DiskB_delta_pos = 0; /* Variable for delta between command and
92                                   actual position of the motor*/
93
94   long int DiskA_delta_pos = 0; /* Variable for delta between command and
95                                   actual position of the arm motor */
96
97   long int New_Position[2] = {0,0}; /* New position to which the motor must go to */
98   long int Actual_Position[2] = {0,0}; /* Actual Position to which the motor must go to */
99
100  long int DiskA_Pos_at_Alignment = 0; /* Position of Disk A when alignment was done */
101  long int DiskB_Pos_at_Alignment = 0;
102
103  int Aligned_Flag = 0;
104  int Target_Reached;
105
106  switch(Number_of_channel)
107  {
108    case 1:
109    {
110      DiskA_Align_switch = 3;
111      DiskB_Align_switch = 5;
112      DiskB_Motor_number = 2;
113      DiskA_Motor_number = 1;
114      break;
115    }
116    case 2:
117    {
118      DiskA_Align_switch = 9;
119      DiskB_Align_switch = 11;
120      DiskB_Motor_number = 4;
121      DiskA_Motor_number = 3;
122      break;
123    }
124    case 3:
125    {
126      DiskA_Align_switch = 15;
127      DiskB_Align_switch = 17;
128      DiskB_Motor_number = 6;
129      DiskA_Motor_number = 5;
130      break;
131    }
132  } /*End of switch case Channel_number */
133
134  /* Read in actual position of motors at this time */
135  Read_Actual_Pos(Address_of_Control,Actual_Position);
136  Read_Actual_Pos(Address_of_Control,Actual_Position);
137
138  printf("\n Disk B motor position = %li ",Actual_Position[0]);
139  printf(" Disk A motor position = %li ",Actual_Position[1]);
140
141  /* Set final position counters to actual position */
142
143  New_Position[0] = Actual_Position[0];
144  New_Position[1] = Actual_Position[1];
145
146  /* Beginning of loop to align system */
147  while (Aligned_Flag == 0)
```

```
148      (
149      /* Test for switch activation */
150      Switch_activated = Check_for_switch_Activation(DiskA_Align_switch);
151
152      printf("\n Moving disks to activate switch %i" ,DiskA_Align_switch);
153
154      /* Move Drum and Arm gears together till switch SW3 has been hit */
155
156      while (Switch_activated != 1)
157      {
158      /*Set up positions for the move command */
159      /* Add delta to drum motor position */
160      New_Position[0] = New_Position[0] + Step_size;
161
162      /* Add delta to arm motor position */
163      New_Position[1] = New_Position[1] - Step_size;
164
165      /* Output new position to the motor controller */
166      Write_Final_pos(Address_of_Control,New_Position);
167
168      /* Start trapezoidal move */
169      Write_to_flag_register (Address_of_Control,0x0808);
170
171      /* Delay reading of motor till it as a chance to move */
172      for (timelaped = 5000; timelaped > 0; timelaped = timelaped - 1);
173
174      /* Check to see if the we have made it to new position or switch
175      has been pressed */
176      do
177      {
178      Switch_activated = Check_for_switch_Activation(DiskA_Align_switch);
179      }
180      while (Switch_activated != 1);
181
182      for (timelaped = 60000; timelaped > 0; timelaped = timelaped - 1);
183
184      Read_Actual_Pos(Address_of_Control,Actual_Position);
185
186      printf("\n Disk B motor position = %li ",Actual_Position[0]);
187      printf(" Disk A motor position = %li ",Actual_Position[1]);
188
189      ) /* End of while (Switch_data != Sw3_detect) */
190
191      printf(" \nSwitch %i limit hit ",DiskA_Align_switch);
192
193      for (timelaped = 200000; timelaped > 0; timelaped = timelaped - 1);
194
195      /* Set all position registers to actual position */
196      Set_Motor_Position(Address_of_Control,Set_to_actual);
197
198      /**** Start reversal of Disk A and Disk B gears till index mark on ***/
199      /**** Disk A motor is found ***/
200
201      /* Read position of arm and drum motor */
202      Read_Actual_Pos(Address_of_Control,Actual_Position);
203
204      printf("\nReverse direction of gear rotation till Disk A index mark is found\n");
205
206      printf("\nDiskB Actual   DiskB Command   DiskA Actual   DiskA Command   Index");
207      printf("\n Position       Position       Position       Position       Switch\n");
208
209      /* Check for index mark */
210      /* Index_Mark_Found = Check_for_Index_Mark(DiskA_Motor_number); */
211
212      Switch_activated = Check_for_switch_Activation(DiskA_Align_switch);
213
214      /* Set delta limit for delta between actual and command position*/
215      Delta_limit = 30;
216
217      New_Position[0] = Actual_Position[0];
218      New_Position[1] = Actual_Position[1];
219
220      while (Switch_activated == 1)
221      {
```

```
222      /* Add delta to arm motor position */
223      New_Position[1] = New_Position[1] + Index_step_size;
224
225      /* Output new position to the motor controller */
226      Write_command_pos(Address_of_Control,New_Position);
227
228      /* Decrease window between actual and command position once switch
229       is activated */
230      if ((Switch_activated == 0) && (Delta_limit > 15))
231          Delta_limit = Delta_limit - 1;
232
233      /* Check to see if the we have made it to new position or switch
234       has been pressed */
235      do
236      {
237
238          /* Read in data to look for index mark and switch activation*/
239          /* Index_Mark_Found = Check_for_Index_Mark(DiskA_Motor_number); */
240          Switch_activated = Check_for_switch_Activation(DiskA_Align_switch);
241
242          /* Check to see if commnaded position has been met */
243          Target_Reached = Reached_Commanded_Pos(Address_of_Control,New_Position,
244                                                  Number_of_channel,Delta_limit);
245
246          if ((Target_Reached == 0) && (Switch_activated == 1))
247          {
248              Read_Actual_Pos(Address_of_Control,Actual_Position);
249              New_Position[1] = Actual_Position[1];
250              Write_command_pos(Address_of_Control,New_Position);
251          }
252
253          Read_Actual_Pos(Address_of_Control,Actual_Position);
254
255          printf(" %10ld %12ld %12ld %12ld %2d %2d\r",
256                Actual_Position[0],New_Position[0],
257                Actual_Position[1],New_Position[1],
258                Index_Mark_Found,Switch_activated);
259
260      }
261
262      while ((Target_Reached == 0 ) && (Switch_activated == 1));
263
264      } /* End of while Index_Mark_Found and
265      && (Switch_activated == 1) */
266
267      printf("\n Disk 'A' gear is aligned !!");
268
269      Read_Actual_Pos(Address_of_Control,Actual_Position);
270
271      /* Store position of disk A */
272      DiskA_Pos_at_Alignment = Actual_Position[1];
273
274      /**** Hold Disk A gear steady and move Disk B gear till *****/
275      /**** switch Sw5 is activated *****/
276
277      printf("\n Moving Disk B to activate switch %i" ,DiskB_Align_switch);
278
279      /* Read Disk A and Disk B motors current position */
280
281      Read_Actual_Pos(Address_of_Control,Actual_Position);
282
283      New_Position[0] = Actual_Position[0];
284      New_Position[1] = DiskA_Pos_at_Alignment;
285
286      /* Check for switch 5 activation */
287      Switch_activated = Check_for_switch_Activation(DiskB_Align_switch);
288
289      while (Switch_activated != 1)
290      {
291          /* Set up positions for the move command */
292          /* Add delta to Disk B motor position */
293          New_Position[0] = New_Position[0] + Step_size;
294
295          /* Add delta to Disk A motor position */
```

```
296     New_Position[1] = DiskA_Pos_at_Alignment;
297
298     /* Output new position to the motor controller */
299     Write_Final_pos(Address_of_Control,New_Position);
300
301     /* Start trapezoidal move */
302     Write_to_flag_registor (Address_of_Control,0x0808);
303
304     /* Check to see if the we have made it to new position or switch
305        has been pressed */
306     do
307     {
308         Switch_activated = Check_for_switch_Activation(DiskB_Align_switch);
309     }
310     while (Switch_activated != 1);
311
312     Read_Actual_Pos(Address_of_Control,Actual_Position);
313
314     printf("\n Disk B motor position = %li ",Actual_Position[0]);
315     printf(" Disk A motor position = %li ",Actual_Position[1]);
316
317     } /* End of while Switch_activated != 1 */
318
319     printf(" \nSwitch %i limit hit ",DiskB_Align_switch);
320
321     for (timelaspd = 200000; timelaspd > 0; timelaspd = timelaspd - 1);
322
323     Set_Motor_Position(Address_of_Control,Set_to_actual);
324
325     /*** Reverse direction of Disk B gears till index mark on Disk B motor ***/
326     /*** is found Disk A gear will be held steady ***/
327
328     printf("\nReverse direction of gear rotation till Disk B index mark is found\n");
329
330     /* Read position of arm and drum motor */
331
332     Read_Actual_Pos(Address_of_Control,Actual_Position);
333
334     New_Position[0] = Actual_Position[0];
335     New_Position[1] = DiskA_Pos_at_Alignment;
336
337     /* Test for index mark and switch activation */
338     /* Index_Mark_Found = Check_for_Index_Mark(DiskB_Motor_number); */
339     Switch_activated = Check_for_switch_Activation(DiskB_Align_switch);
340
341     /* Set delta limit for delta between actual and command position*/
342     Delta_limit = 30;
343
344     printf("\nDiskB Actual    DiskB Command    DiskA Actual    DiskA Command    Index(/");
345     printf("\n Position          Position          Position          Position          Switch\n");
346
347     while (Switch_activated == 1)
348     {
349         /* Add delta to drum motor position */
350         New_Position[0] = New_Position[0] - Index_step_size;
351
352         /* Add delta to arm motor position */
353         New_Position[1] = DiskA_Pos_at_Alignment;
354
355         /* Output new position to the motor controller */
356         Write_command_pos(Address_of_Control,New_Position);
357
358         /* Decrease window between actual and command position once switch
359            is activated */
360         if ((Switch_activated == 0) && (Delta_limit > 15))
361             Delta_limit = Delta_limit - 1;
362
363         /* Check to see if the we have made it to new position or switch
364            has been pressed */
365         do
366         {
367
368             /* Read in data to look for index mark */
369             /* Index_Mark_Found = Check_for_Index_Mark(DiskB_Motor_number); */
```

```
370      Switch_activated = Check_for_switch_Activation(DiskB_Align_switch);
371
372      /* Check to see if commnaded position has been met */
373      Target_Reached = Reached_Commanded_Pos(Address_of_Control,New_Position,
374      Number_of_channel,Delta_limit);
375
376
377      /* Resend position command if delta becomes to large and
378      switch is not activated */
379
380      if ((Target_Reached == 0) && (Switch_activated == 1))
381      {
382          Read_Actual_Pos(Address_of_Control,Actual_Position);
383          New_Position[0] = Actual_Position[0];
384          Write_command_pos(Address_of_Control,New_Position);
385      }
386
387      Read_Actual_Pos(Address_of_Control,Actual_Position);
388
389      printf(" %10ld %12ld %12ld %12ld %2d %2d \r",
390      Actual_Position[0],New_Position[0],
391      Actual_Position[1],New_Position[1],
392      Index_Mark_Found, Switch_activated);
393
394  }
395  while ((Target_Reached == 0) && (Switch_activated == 1));
396
397
398  } /* End of while Index_Mark_Found != 0 */
399
400  printf("\n Disk 'B' gear is aligned !! \n");
401
402  /* Set up for command to go to alingment for both motors to index mark */
403
404  New_Position[0] = Actual_Position[0];
405  DiskB_Pos_at_Alignment = Actual_Position[0];
406  New_Position[1] = DiskA_Pos_at_Alignment;
407  Write_command_pos(Address_of_Control,New_Position);
408
409  Aligned_Flag = 1;
410
411  } /* End of while aligned = 0 */
412
413  Delta_limit = 15;
414
415  /***** Move arm back to home position *****/
416
417  /* Reset all position registers to zero */
418  Set_Motor_Position(Address_of_Control,Reset_to_zero);
419
420  New_Position[0] = -25000;
421  New_Position[1] = 0;
422
423  /* Initate movement of motor to swing over arm */
424  Write_Final_pos(Address_of_Control,New_Position);
425  Write_to_flag_registor (Address_of_Control,0x0808);
426
427  Target_Reached = 0;
428
429  while (Target_Reached != 1)
430  {
431      /* Check to see if commnaded position has been met */
432      Target_Reached = Reached_Commanded_Pos(Address_of_Control,New_Position,
433      Number_of_channel,Delta_limit);
434  }
435
436  /*Store data as required and move to home position */
437  if (Store_Flag == 1)
438  {
439      New_Position[0] = -DiskB_Pos_at_Alignment;
440      New_Position[1] = -DiskA_Pos_at_Alignment;
441
442      /* Read in data with present values for alignment */
443
```

```
444     Read_Align_data();
445
446     Align_Position[0][Number_of_channel-1] = DiskB_Pos_at_Alignment;
447     Align_Position[1][Number_of_channel-1] = DiskA_Pos_at_Alignment;
448
449     /* Write out new data for alignment */
450     Write_Align_data(); /* Write out new alignment data */
451 }
452     else
453
454 {
455     Read_Align_data(); /* Read in data for alignment purposes */
456
457     /* Compute position for Disk B motor for arm only move */
458     New_Position[0] = -Align_Position[0][Number_of_channel-1];
459     New_Position[1] = -Align_Position[1][Number_of_channel-1];
460 }
461
462 /* Initiate movement of motor */
463 Write_Final_pos(Address_of_Control,New_Position);
464 Write_to_flag_registor (Address_of_Control,0x0808);
465
466     Target_Reached = 0;
467
468     while (Target_Reached != 1)
469 {
470     /* Check to see if commnaded position has been met */
471     Target_Reached = Reached_Commanded_Pos(Address_of_Control,New_Position,
472     Number_of_channel,Delta_limit);
473     printf("\rWaiting for channel to reach home position");
474
475     for (timelaspded = 10000; timelaspded > 0; timelaspded = timelaspded - 1);
476 }
477
478 /* Wait for motors to stop running */
479 for (timelaspded = 60000; timelaspded > 0; timelaspded = timelaspded - 1);
480
481 /* Reset motor to 0,0 position */
482 Set_Motor_Position(Address_of_Control,Reset_to_zero);
483
484 /* Store new starting position after alignment */
485 Starting_Position[0][Number_of_channel-1] = 0;
486 Starting_Position[1][Number_of_channel-1] = 0;
487
488 } /* End of alignment function */
489
490
491 /* Function will be used to rotate platter through is paces */
492 int Exercise_Channel(unsigned Control_Address,int chan_number)
493 {
494     int long alpha_move = -40;
495     int long delta_move = 100;
496     int long New_Pos[2] = {0,0};
497     int long Send_Pos[2] = {0,0};
498     int long Actual_Pos[2] = {0,0};
499     int switch_number = 3;
500     int count;
501     int Activated_switch;
502     Exit_requested = 0;
503
504     while (Exit_requested == 0)
505     {
506         for (count = 1; count <= 4 ; count = count + 1)
507         {
508             switch (count)
509             {
510                 case 1:
511                 {
512                     if (chan_number == 1) switch_number = 3;
513                     else if (chan_number == 2) switch_number = 9;
514                     else if (chan_number == 3) switch_number = 15;
515                     alpha_move = 0;
516                     delta_move = +500;
517                     break;
```

```
518     )
519     case 2:
520     (
521         if (chan_number == 1) switch_number = 4;
522         else if (chan_number == 2) switch_number = 10;
523         else if (chan_number == 3) switch_number = 16;
524         alpha_move = 0;
525         delta_move = -500;
526         break;
527     )
528     case 3:
529     (
530         if (chan_number == 1) switch_number = 5;
531         else if (chan_number == 2) switch_number = 11;
532         else if (chan_number == 3) switch_number = 17;
533         alpha_move = +100;
534         delta_move = 40;
535         break;
536     )
537
538     case 4:
539     (
540         if (chan_number == 1) switch_number = 6;
541         else if (chan_number == 2) switch_number = 12;
542         else if (chan_number == 3) switch_number = 18;
543         alpha_move = -100;
544         delta_move = 0;
545         break;
546     )
547     } /* End of switch count case*/
548
549     Set_Motor_Position(Control_Address,Set_to_actual);
550
551     /* Delay reading of motor till it as a chance to move */
552     for (timelaspd = 50000; timelaspd > 0; timelaspd = timelaspd - 1);
553
554     Read_Actual_Pos(Control_Address,Actual_Pos);
555
556     /*Set up positions for the move command */
557     /* Add delta to Disk B and Disk A motor position */
558     New_Pos[0] = (alpha_move * 769) + (1967 * delta_move) + Actual_Pos[0];
559     New_Pos[1] = Actual_Pos[1] - (1967 * delta_move);
560
561     /* Output new position to the motor controller */
562     Write_Final_pos(Control_Address,New_Pos);
563
564     /* Start trapezoidal move */
565     Write_to_flag_register (Control_Address,0x0808);
566
567     /* Check to see if the we have made it to new position or switch
568        has been pressed */
569     do
570     (
571         if (kbhit())
572         (
573             Exit_requested = 1;
574             break;
575         )
576         Activated_switch = Check_for_switch_Activation(switch_number);
577     )
578     while (Activated_switch != 1);
579
580     /* Exit for loop if key has been pressed */
581     if (Exit_requested == 1) break;
582
583     /* Delay reading of motor till it as a chance to move */
584     for (timelaspd = 50000; timelaspd > 0; timelaspd = timelaspd - 1);
585
586     if (count == 4) /* Move arm to center position */
587     (
588         Set_Motor_Position(Control_Address,Set_to_actual);
589
590         /* Delay reading of motor till it as a chance to move */
591         for (timelaspd = 50000; timelaspd > 0; timelaspd = timelaspd - 1);
```



```
592
593     Read_Actual_Pos(Control_Address,Actual_Pos);
594
595     delta_move = 0;
596     alpha_move = 40;
597
598     /*Set up positions for the move command */
599     /* Add delta to Disk B and Disk A motor position */
600
601     New_Pos[0] = (alpha_move * 769) + (1967 * delta_move) + Actual_Pos[0];
602     New_Pos[1] = Actual_Pos[1] - (1967 * delta_move);
603
604     Send_Pos[0] = New_Pos[0];
605     Send_Pos[1] = New_Pos[1];
606
607     /* Output new position to the motor controller and start move*/
608     Write_Final_pos(Control_Address,Send_Pos);
609     Write_to_flag_register (Control_Address,0x0808);
610
611     /* Wait till arm makes its way back to home position */
612     while (labs(Actual_Pos[0] - New_Pos[0]) > 25)
613     {
614         /* Delay to give motor a chance to respond to trapizodal move */
615         for (timelaped = 5000; timelaped > 0; timelaped = timelaped - 1);
616         Read_Actual_Pos(Control_Address,Actual_Pos);
617     }
618
619 }
620
621 } /* End of for loop */
622
623     } /* End of while 1*/
624
625     /* Stop motors and reset all positions counters to actual position */
626     Set_Motor_Position(Control_Address,Set_to_actual);
627
628 }/* End of function */
629
630
631 void Run_Back_Lash_Test(unsigned Address_Motor,int Port_C_Data,int Numb_Chans,
632                        int Number_of_Motor)
633 {
634     int Frame_Count = 0;
635     double Alpha;
636
637     /* Print out title indicating test to be run */
638     if (Number_of_Motor == 1)
639         fprintf(Manual_test_ptr,"\nBacklash test for Disk B (U_error");
640     else if (Number_of_Motor == 2)
641         fprintf(Manual_test_ptr,"\nBacklash test for Disk A ");
642     else if (Number_of_Motor == 3)
643         fprintf(Manual_test_ptr,"\nBacklash test for Disk A & B moving together (V_error)");
644     else if (Number_of_Motor == 4)
645         fprintf(Manual_test_ptr,"\nBacklash test for Disk A & B moving a part");
646
647
648
649     /* Print header file for data */
650     fprintf(Manual_test_ptr,"\nTimeTag \tSig Lev\tU_Error\tV_Error");
651     fprintf(Manual_test_ptr,"\tActB\tDiskBC\t ActA\tDiskAC");
652
653     Set_Motor_Position(Address_Motor,Set_to_actual);
654
655     /* Read data from system file on present position */
656     Read_Current_System_data();
657
658     DiskAstep = Current_Position[1][Numb_Chans - 1];
659     DiskBstep = Current_Position[0][Numb_Chans - 1];
660
661     Compute_AZ_EL (DiskAstep,DiskBstep,0,El_Az,Alpha_Delta);
662
663     Alpha = Alpha_Delta[0]/57.29577951;
664
665     Compute_Open_Loop_Commands(Address_Motor,Number_of_Motor,Alpha); /* Compute commands for open loop
```

```

666
667     Port_C_Data = Port_C_Data | Sync_Pulse_Enable[Numb_Chann - 1];
668     outp(Port_C_Address,Port_C_Data);
669
670     ComFlushRx();
671
672     while (Frame_Count <= Number_of_Frames)
673
674     {
675         while (ComLenRx() < BLK_SZ); /* Wait for data to come over */
676
677         Read_Actual_Pos(Address_Motor,Act_pos);
678         Step_Position[0] = Motor_Pos[0][Frame_Count + 1];
679         Step_Position[1] = Motor_Pos[1][Frame_Count + 1];
680
681         Write_Final_pos(Address_Motor,Step_Position);
682         Write_to_flag_registor (Address_Motor,0x0808);
683
684         Read_Channel_Data(&Test[Numb_Chann - 1]);
685
686         if (Act_pos[0] >= 0x7fffff)
687             Act_pos[0] -= 0xfffff;
688
689         if (Act_pos[1] >= 0x7fffff)
690             Act_pos[1] -= 0xfffff;
691
692         fprintf(Manual_test_ptr," \n%u, %2.3f, \t %1.5f, \t%1.5f, \t%li,\t %li,\t %li,\t%li",
693             Test[Numb_Chann - 1].Time_tag,Test[Numb_Chann - 1].Signal_strength/204.7,
694             Test[Numb_Chann - 1].U_error,Test[Numb_Chann - 1].V_error,
695             Act_pos[0],Motor_Pos[0][Frame_Count],
696             Act_pos[1],Motor_Pos[1][Frame_Count]);
697
698         Frame_Count++;
699     } /* End of while loop */
700
701     Port_C_Data = Port_C_Data & Sync_Pulse_Disable[Numb_Chann - 1];
702     outp(Port_C_Address,Port_C_Data);
703
704 } /* End of function Run_Back_Lash_Test */
705
706
707
708
709 void Run_Step_Test(unsigned Add_of_Motor,int C_number,int Step_value,int Option)
710 {
711     int long Act_step_pos_array_B[1000];
712     int long Act_step_pos_array_A[1000];
713     int Sample_total = 1000;
714     int Number_of_samples = 0;
715     int Array_count = 0;
716     unsigned AD_data[1000];
717     int AD_dataA,AD_dataB;
718     unsigned UAD_dataA,UAD_dataB;
719
720     Read_Actual_Pos(Add_of_Motor,Step_Position);
721
722     if ((Option == 1) || (Option == 5))
723     {
724         Step_Position[0] = Step_value + Step_Position[0]; /* Assign step value for disk b motor */
725         Step_Position[1] = 0 + Step_Position[1];
726     }
727     else if ((Option == 2) || (Option == 6))
728     {
729         Step_Position[1] = Step_value + Step_Position[1]; /* Assign step value for disk b motor */
730         Step_Position[0] = 0 + Step_Position[0];
731     }
732     else if ((Option == 3) || (Option == 7))
733     {
734         Step_Position[1] = - Step_value + Step_Position[1]; /* Assign step value for disk b motor */
735         Step_Position[0] = Step_value + Step_Position[0]; /* to run in same direction */
736     }
737     else if ((Option == 4) || (Option == 8))
738     {
739         Step_Position[1] = Step_value + Step_Position[1]; /* Assign step value for disk b motor */

```

```

740     Step_Position[0] = Step_value + Step_Position[0];
741 }
742
743     if ((Option >= 1) && (Option <= 4))
744         Write_command_pos(Add_of_Motor, Step_Position);
745     else
746     {
747         Write_Final_pos(Add_of_Motor, Step_Position);
748         Write_to_flag_registor (Add_of_Motor, 0x0808);
749     }
750
751     while (Number_of_samples < Sample_total)
752     {
753         AD_data[Number_of_samples] = Read_8bit_port(Add_of_Motor);
754         Read_Actual_Pos(Add_of_Motor, Act_pos);
755         Act_step_pos_array_B[Number_of_samples] = Act_pos[0];
756         Act_step_pos_array_A[Number_of_samples] = Act_pos[1];
757         Number_of_samples++;
758     }
759
760     fprintf(Manual_test_ptr, "\nNumber of samples taken = %i", Number_of_samples);
761
762     /* Store sampled data */
763     for (Array_count = 0; Array_count < Number_of_samples; Array_count++)
764     {
765         UAD_dataB = AD_data[Array_count] & 0xFF;
766         AD_dataB = (UAD_dataB) - 128;
767         UAD_dataA = AD_data[Array_count] & 0xFF00;
768         UAD_dataA = UAD_dataA >> 8;
769         AD_dataA = UAD_dataA - 128;
770
771         fprintf(Manual_test_ptr, "\n%d", Act_step_pos_array_B[Array_count]);
772         fprintf(Manual_test_ptr, " %d ", AD_dataB);
773         fprintf(Manual_test_ptr, " %d", Act_step_pos_array_A[Array_count]);
774         fprintf(Manual_test_ptr, " %d ", AD_dataA);
775     }
776 }
777
778 void Run_Eccen_Test(unsigned Motor_Address, int CPort, int Number_of_Channel)
779 {
780     int DA_data, AD_dataB, AD_dataA;
781     unsigned UAD_dataB, UAD_dataA;
782     long int Present_Position[2];
783     long int Command_Pos[2];
784     int Left_Limit[3] = {4, 10, 16};
785     int Right_Limit[3] = {3, 9, 15};
786     int Limit_Hit = 0;
787
788     fprintf(Manual_test_ptr, "DiskBApos D/A DiskAApos D/A");
789
790     /* Set up position to move to designated limit */
791     Command_Pos[0] = 1000000;
792     Command_Pos[1] = -1000000;
793
794     Read_Actual_Pos(Motor_Address, Present_Position);
795     Command_Pos[0] = Present_Position[0] + Command_Pos[0];
796     Command_Pos[1] = Present_Position[1] + Command_Pos[1];
797
798     /* Output new position to the motor controller and start move */
799     Write_Final_pos(Motor_Address, Command_Pos);
800     Write_to_flag_registor (Motor_Address, 0x0808);
801
802     /* Wait till first limit has been reached */
803     while (Limit_Hit == 0)
804     {
805         Limit_Hit = Check_for_switch_Activation(Right_Limit[Number_of_Channel - 1]);
806     }
807
808     Set_Motor_Position(Motor_Address, Set_to_actual);
809
810     /* Set up position to move to designated limit */
811     Command_Pos[0] = -1000000;
812
813

```

```

814 Command_Pos[1] = +1000000;
815
816 Read_Actual_Pos(Motor_Address,Present_Position);
817 Command_Pos[0] = Present_Position[0] + Command_Pos[0];
818 Command_Pos[1] = Present_Position[1] + Command_Pos[1];
819
820 /* Output new position to the motor controller and start move*/
821 Write_Final_pos(Motor_Address,Command_Pos);
822 Write_to_flag_register (Motor_Address,0x0808);
823
824 /* Enable receiver electronics for timing purposes */
825 CPort = CPort | Sync_Pulse_Enable[Number_of_Channel - 1];
826 outp(Port_C_Address,CPort);
827
828 ComFlushRx();
829 Limit_Hit = 0;
830
831 while (Limit_Hit == 0)
832
833 {
834 while (ComLenRx() < BLK_SZ); /* Wait for data to come over */
835
836 ComFlushRx(); /* Flush out receive buffer to avoid overflow */
837
838 DA_data = Read_8bit_port(Motor_Address);
839 Read_Actual_Pos(Motor_Address,Present_Position);
840
841 if (Present_Position[1] > 0x7fffff)
842 Present_Position[1] -= 0xfffff;
843
844 if (Present_Position[0] > 0x7fffff)
845 Present_Position[0] -= 0xfffff;
846
847 UAD_dataB = DA_data & 0x00FF;
848 AD_dataB = UAD_dataB - 128;
849 UAD_dataA = DA_data & 0xFF00;
850 UAD_dataA = UAD_dataA >> 8;
851 AD_dataA = UAD_dataA - 128;
852
853 /* Output data to file for reveive */
854 fprintf(Manual_test_ptr,"\n%10ld",Present_Position[0]);
855 fprintf(Manual_test_ptr," %5d ",AD_dataB);
856 fprintf(Manual_test_ptr," %10ld",Present_Position[1]);
857 fprintf(Manual_test_ptr," %5d ",AD_dataA);
858
859 Limit_Hit = Check_for_switch_Activation(Left_Limit[Number_of_Channel - 1]);
860 }
861
862 /* Disable sync pulse to receiver electronics */
863 CPort = CPort & Sync_Pulse_Disable[Number_of_Channel - 1];
864 outp(Port_C_Address,CPort);
865
866 } /* Function Compute_Open_Loop Command */
867
868 void Compute_Open_Loop_Commands(unsigned Channel_Address,int Motor_Number,
869 double Alpha_arm)
870 {
871 int Frame_count = 0;
872 double Int_Motor_PosA = 0;
873 double Int_Motor_PosB = 0;
874
875 if (Motor_Number == 1) /* Move only disk b motor */
876 {
877 Step_A = 0;
878 Step_B = 1.0;
879 }
880 else if (Motor_Number == 2) /* Move only disk a motor: */
881 {
882 Step_A = -1.0;
883 Step_B = 0;
884 }
885
886 else if (Motor_Number == 3) /* Move only disk A & B motor:
887 disk will run in same direction*/

```

```
888 {
889     Step_A = 2.729 - sin((double)(Alpha_arm/2.0) - (Pi/4.0));
890     Step_B = 2.729;
891 }
892
893     else if (Motor_Number == 4) /* Move only disk A & B motor:
894                                disk will run in same direction*/
895     {
896         Step_A = -1.0;
897         Step_B = 1.0;
898     }
899
900     /* Read in actual position of the arm and drum motors for offset */
901     Read_Actual_Pos(Channel_Address,Act_pos);
902
903     Motor_Pos[0][Frame_count] = Act_pos[0];
904     Motor_Pos[1][Frame_count] = Act_pos[1];
905
906     Int_Motor_PosB =(double)Act_pos[0];
907     Int_Motor_PosA =(double)Act_pos[1];
908
909
910     Frame_count = Frame_count + 1;
911
912     /* Check to see if range of frame counts is such that both motors will
913        be moved together */
914
915     while (Frame_count <= Number_of_Frames)
916     {
917         if (Frame_count <= Range)
918         {
919
920             Int_Motor_PosB = Int_Motor_PosB + Step_B;
921             Motor_Pos[0][Frame_count] = (long int)Int_Motor_PosB;
922             Int_Motor_PosA = Int_Motor_PosA - Step_A;
923             Motor_Pos[1][Frame_count] = (long int)Int_Motor_PosA;
924
925         }
926
927         else if ((Frame_count <= (3*Range)) && (Frame_count > Range))
928         {
929             Int_Motor_PosB = Int_Motor_PosB - Step_B;
930             Motor_Pos[0][Frame_count] = (long int)Int_Motor_PosB;
931             Int_Motor_PosA = Int_Motor_PosA + Step_A;
932             Motor_Pos[1][Frame_count] = (long int)Int_Motor_PosA;
933         }
934
935         else if ((Frame_count <= (4*Range)) && (Frame_count > (3*Range)))
936         {
937             Int_Motor_PosB = Int_Motor_PosB + Step_B;
938             Motor_Pos[0][Frame_count] = (long int)Int_Motor_PosB;
939             Int_Motor_PosA = Int_Motor_PosA - Step_A;
940             Motor_Pos[1][Frame_count] = (long int)Int_Motor_PosA;
941         }
942
943         else if ((Frame_count <= (5*Range)) && (Frame_count > (4*Range)))
944         {
945             Int_Motor_PosB = Int_Motor_PosB - Step_B;
946             Motor_Pos[0][Frame_count] = (long int)Int_Motor_PosB;
947             Int_Motor_PosA = Int_Motor_PosA - Step_A;
948             Motor_Pos[1][Frame_count] = (long int)Int_Motor_PosA;
949         }
950         else if ((Frame_count <= (7*Range)) && (Frame_count > (5*Range)))
951         {
952             Int_Motor_PosB = Int_Motor_PosB + Step_B;
953             Motor_Pos[0][Frame_count] = (long int)Int_Motor_PosB;
954             Int_Motor_PosA = Int_Motor_PosA + Step_A;
955             Motor_Pos[1][Frame_count] = (long int)Int_Motor_PosA;
956         }
957         else if ((Frame_count <= (8*Range)) && (Frame_count > (7*Range)))
958         {
959             Int_Motor_PosB = Int_Motor_PosB - Step_B;
960             Motor_Pos[0][Frame_count] = (long int)Int_Motor_PosB;
961             Int_Motor_PosA = Int_Motor_PosA - Step_A;
962             Motor_Pos[1][Frame_count] = (long int)Int_Motor_PosA;
```

```
962     }
963
964         Frame_count = Frame_count + 1;
965     }
966
967     Motor_Pos[0][Frame_count] = Motor_Pos[0][0];
968     Motor_Pos[1][Frame_count] = Motor_Pos[1][0];
969
970 } /* End of Compute_Open_Loop_Commands function */
971
972
973
974
975
976
```



```

74  int Max_Phase_Adv_Address = 0x1f1f; /* Address to command upper limit of
75                                     phase advance */
76
77  int Filter_Zero_Address = 0x2020; /* Address to set up value for zero in
78                                     digital filter */
79
80  int Filter_Pole_Address = 0x2121; /* Address to set up value for pole in
81                                     digital filter */
82
83  int Filter_Gain_Address = 0x2222; /* Address to set gain of the digital filter */
84
85  int Comm_Velocity1_Address = 0x2323; /* Address to input LSB of command velocity */
86
87  int Comm_Velocity2_Address = 0x2424; /* Address to input MSB of command velocity */
88
89  int Accel_LSB_Address = 0x2626; /* Address to input LSB of command acceleration */
90
91  int Accel_MSB_Address = 0x2727; /* Address to input MSB of command acceleration */
92
93  int Max_Velocity_Address = 0x2828; /* Address to max velocity in the trapezoidal
94                                     profile */
95
96  int Final_Pos1_Address = 0x2929; /* Address (LSB) to write or read the position
97                                     at which the motor is commanded to go to */
98
99  int Final_Pos2_Address = 0x2a2a; /* Address to write or read the position
100                                    at which the motor is commanded to go to */
101
102  int Final_Pos3_Address = 0x2b2b; /* Address (MSB) to write or read the position
103                                    at which the motor is commanded to go to */
104
105  int Actual_Vel1_Address = 0x3434; /* Address (LSB) to read actual velocity of motor */
106
107  int Actual_Vel2_Adresss = 0x3535; /* Address (MSB) to read actual velocity of motor */
108
109  /* Listed below is all the functions in the program */
110
111  int Read_Filter_Zero(unsigned Motor_Address);
112  int Display_Main_Menu(void);
113  void Wait_for_key_Press(void);
114  void Process_Main_Option(int Key_pressed);
115  int Display_Main_Menu(void);
116  void Write_Filter_Zero(unsigned Motor_Address,int Filter_Zero);
117  void Write_Filter_Pole(unsigned Motor_Address,int Filter_Pole);
118  void Write_Filter_Gain(unsigned Motor_Address,int Filter_Gain);
119  int Read_Filter_Zero(unsigned Motor_Address);
120  int Read_Filter_Pole(unsigned Motor_Address);
121  int Read_Filter_Gain(unsigned Motor_Address);
122  void Write_to_8bit_port(unsigned Motor_Address,int Digital_word);
123  void Write_to_PWM_port(unsigned Motor_Address,int PWM_data);
124  void Write_to_sampler_timer(unsigned Motor_Address, int Sampler_data);
125  unsigned Read_status_reg(unsigned Motor_Address);
126  void Write_Max_Velocity(unsigned Motor_Address,int MaxVelocity);
127  void Write_Max_Accel(unsigned Motor_Address, int *Max_Acc_data);
128  void Write_to_program_counter (unsigned Motor_Address,int Program_counter);
129  void Write_Final_pos(unsigned Motor_Address,long int *Final_data);
130  Read_Actual_Pos(unsigned Motor_Address, long int * Actual_pos);
131  void Write_to_flag_register (unsigned Motor_Address,int Flagword);
132  void Reset_Actual_Position(unsigned Motor_Address,int Reset_Pos);
133  Read_Final_Pos(unsigned Motor_Address, long int *Final_pos);
134  int Read_Max_Accel(unsigned Motor_Address, int *Motor_Acceleration);
135  int Read_Max_Velocity(unsigned Motor_Address);
136  Read_Command_Pos(unsigned Motor_Address,long int *Command_pos);
137  void Write_command_pos(unsigned Motor_Address,long int *Position_data);
138
139
140  /* The following function will be used to write to both the drum and arm
141     motor controller at the same time */
142
143
144  void Write_to_motor_controller(unsigned Address, int Dataword, int Register)
145  (
146     outpw (Address , Register); /*Generate an Ale signal and send over the
147                                   register address that we want to write

```



```
148             to */
149
150     outpw ((Address + 2), Dataword); /*Generate chip select and output data */
151 }
152
153
154 /* The following function will be used to read data from the both
155    the drum and motor controllers */
156
157 int Read_from_motor_controller(unsigned Address, int Registor)
158 {
159     int Dumb_data, Controller_data;
160
161     outpw(Address, Registor); /*Generate an Ale signal and send over the
162                                register address that we want to write
163                                to */
164
165
166     Dumb_data = inpw(Address + 2); /*Generate chip select to motor controller */
167
168     Controller_data = inpw(Address + 4 ); /* Generate OE signal and read data */
169
170     return Controller_data;
171 }
172
173
174 /* The following function will write to the flag registor */
175
176 void Write_to_flag_registor (unsigned Motor_Address, int Flagword)
177 {
178     Write_to_motor_controller(Motor_Address, Flagword, Flag_Reg_Address);
179 }
180
181
182 /* The following function will write to the program counter*/
183
184 void Write_to_program_counter (unsigned Motor_Address, int Program_counter)
185 {
186     Write_to_motor_controller(Motor_Address, Program_counter, Prog_Counter_Address);
187 }
188
189
190 /* The following function will write to the status registor*/
191
192 void Write_to_status_reg(unsigned Motor_Address, int Statusword)
193 {
194     Write_to_motor_controller(Motor_Address, Statusword, Status_Reg_Address);
195 }
196
197
198 /* The following function will read the status registor*/
199
200 unsigned Read_status_reg(unsigned Motor_Address)
201 {
202     unsigned Status;
203     Status = Read_from_motor_controller(Motor_Address, Status_Reg_Address);
204     return Status;
205 }
206
207
208 /* The following function will write to the 8 bit port*/
209
210 void Write_to_8bit_port(unsigned Motor_Address, int Digital_word)
211 {
212     Write_to_motor_controller(Motor_Address, Digital_word, AD_Motor_Com_Address);
213 }
214
215
216 /* The following function will read the data in the 8 bit port*/
217
218 int Read_8bit_port(unsigned Motor_Address)
219 {
220     int Port_data;
221     Port_data = Read_from_motor_controller(Motor_Address, AD_Motor_Com_Address);
```

```
222     return Port_data;
223
224 }
225
226 /* The following function will write data to the PWM port*/
227
228 void Write_to_PWM_port(unsigned Motor_Address,int PWM_data)
229 {
230     Write_to_motor_controller(Motor_Address,PWM_data,Pwm_Motor_Com_Address );
231
232 }
233
234 /* The following function will read the data from the pwm command register*/
235
236 int Read_PWM_port(unsigned Motor_Address)
237 {
238     int PWM_Port_data;
239     PWM_Port_data = Read_from_motor_controller(Motor_Address,Pwm_Motor_Com_Address);
240     return PWM_Port_data;
241 }
242
243
244
245 /* The following function will write data to the Command position register*/
246
247 void Write_command_pos(unsigned Motor_Address,long int *Position_data)
248 {
249     long int New_Pos_Data[2];
250     int Pos_data = 0;
251
252     /* Invert the sign of the command if channel 1 is written to */
253     if (Motor_Address == Motor_Pair1_Address)
254     {
255         New_Pos_Data[0] = Position_data[0] * -1;
256         New_Pos_Data[1] = Position_data[1] * -1;
257     }
258     else
259     {
260         New_Pos_Data[0] = Position_data[0];
261         New_Pos_Data[1] = Position_data[1];
262     }
263
264     /* Output the most significant byte to the motor controller */
265     Pos_data = (((New_Pos_Data[0] >> 16) & 0xff) | ((New_Pos_Data[1] >> 8) & 0xff00));
266     Write_to_motor_controller(Motor_Address,Pos_data,Command_Pos3_Address);
267
268     /* Output the middle byte to the motor controller */
269     Pos_data = ((New_Pos_Data[0] >> 8) & 0xff) | (New_Pos_Data[1] & 0xff00);
270     Write_to_motor_controller(Motor_Address,Pos_data,Command_Pos2_Address);
271
272     /* Output the least significant byte to the motor controller */
273     Pos_data = ((New_Pos_Data[0] & 0xff) | (New_Pos_Data[1] << 8) & 0xff00);
274     Write_to_motor_controller(Motor_Address,Pos_data,Command_Pos1_Address);
275
276 }
277
278
279
280 /* The following function will read the command motor position register*/
281
282 Read_Command_Pos(unsigned Motor_Address,long int *Command_pos)
283 {
284
285     int Com_data1,Com_data2,Com_data3;
286
287     /* Read the most significant byte of the command position from the motor controller */
288     Com_data3 = Read_from_motor_controller(Motor_Address,Command_Pos3_Address);
289
290     /* Read the middle byte of the command position from the motor controller */
291     Com_data2 = Read_from_motor_controller(Motor_Address,Command_Pos2_Address);
292
293     /* Read the least significant byte of the command position from the motor controller */
294     Com_data1 = Read_from_motor_controller(Motor_Address,Command_Pos1_Address);
295 }
```

```
296
297 /* Determine actual pos for drum motor */
298 Command_pos[0] = Com_data3 & 0xff;
299 Command_pos[0] = (Command_pos[0] << 8) | (Com_data2 & 0xff);
300 Command_pos[0] = (Command_pos[0] << 8) | (Com_data1 & 0xff);
301
302 /* Determine actual pos for arm motor */
303 Command_pos[1] = Com_data3 & 0xff00;
304 Command_pos[1] = (Command_pos[1] << 8) | (Com_data2 & 0xff00);
305 Command_pos[1] = (Command_pos[1] | ((Com_data1 & 0xff00) >> 8));
306
307 if (Motor_Address == Motor_Pair1_Address)
308 {
309     Command_pos[0] = Command_pos[0] * -1;
310     Command_pos[1] = Command_pos[1] * -1;
311 }
312
313
314 }
315
316
317
318 /* The following function will write data to the Sampler timer register*/
319
320 void Write_to_sampler_timer(unsigned Motor_Address, int Sampler_data)
321 {
322     /* Output the timer value to the motor controller */
323     Write_to_motor_controller(Motor_Address, Sampler_data, Sampler_Timer_Address);
324 }
325
326
327
328
329 /* The following function will read the actual motor position register*/
330
331 Read_Actual_Pos(unsigned Motor_Address, long int * Actual_pos)
332 {
333     int Actual_data1, Actual_data2, Actual_data3;
334
335     /* Read the least significant byte of the command position from the motor controller */
336     Actual_data1 = Read_from_motor_controller(Motor_Address, Actual_Pos1_Address);
337
338     /* Read the middle byte of the command position from the motor controller */
339     Actual_data2 = Read_from_motor_controller(Motor_Address, Actual_Pos2_Address);
340
341     /* Read the most significant byte of the command position from the motor controller */
342     Actual_data3 = Read_from_motor_controller(Motor_Address, Actual_Pos3_Address);
343
344     /* Shift and or the data read from motor controller and store in a 21 byte
345     word */
346
347     /* Determine actual pos for drum motor */
348     Actual_pos[0] = Actual_data3 & 0xff;
349     Actual_pos[0] = (Actual_pos[0] << 8) | (Actual_data2 & 0xff);
350     Actual_pos[0] = (Actual_pos[0] << 8) | (Actual_data1 & 0xff);
351
352     if (Actual_pos[0] > 0x7fffff)
353         Actual_pos[0] -= 0x7fffff;
354
355     /* Determine actual pos for arm motor */
356     Actual_pos[1] = Actual_data3 & 0xff00;
357     Actual_pos[1] = (Actual_pos[1] << 8) | (Actual_data2 & 0xff00);
358     Actual_pos[1] = (Actual_pos[1] | ((Actual_data1 & 0xff00) >> 8));
359
360     if (Actual_pos[1] > 0x7fffff)
361         Actual_pos[1] -= 0x7fffff;
362
363     if (Motor_Address == Motor_Pair1_Address)
364     {
365         Actual_pos[0] = Actual_pos[0] * -1;
366         Actual_pos[1] = Actual_pos[1] * -1;
367     }
368 }
369
```

```
370
371     }
372
373     /* The following function reset the actual position register*/
374
375     void Reset_Actual_Position(unsigned Motor_Address,int Reset_Pos)
376
377     {
378         /* Write to register 13H to reset actual position counter */
379         Write_to_motor_controller(Motor_Address,Reset_Pos,Actual_Pos2_Address);
380     }
381
382
383
384     /* The following function writes the zero of the digital filter to the
385        to the motor controller */
386
387     void Write_Filter_Zero(unsigned Motor_Address,int Filter_Zero)
388
389     {
390         Write_to_motor_controller(Motor_Address,Filter_Zero,Filter_Zero_Address);
391     }
392
393
394
395     /* The following function will read value of the register that contains the
396        zero of the digital filter*/
397
398     int Read_Filter_Zero(unsigned Motor_Address)
399
400     {
401         int Zero;
402
403         Zero = Read_from_motor_controller(Motor_Address,Filter_Zero_Address);
404         return Zero;
405     }
406
407
408     /* The following function writes the pole of the digital filter to the
409        to the motor controller */
410
411     void Write_Filter_Pole(unsigned Motor_Address,int Filter_Pole)
412
413     {
414         Write_to_motor_controller(Motor_Address,Filter_Pole,Filter_Pole_Address);
415     }
416
417
418
419     /* The following function will read value of the register that contains the
420        pole of the digital filter*/
421
422     int Read_Filter_Pole(unsigned Motor_Address)
423
424     {
425         int Pole;
426
427         Pole = Read_from_motor_controller(Motor_Address,Filter_Pole_Address);
428         return Pole;
429     }
430
431     /* The following function writes the Gain of the digital filter to the
432        to the motor controller */
433
434     void Write_Filter_Gain(unsigned Motor_Address,int Filter_Gain)
435
436     {
437         Write_to_motor_controller(Motor_Address,Filter_Gain,Filter_Gain_Address);
438     }
439
440
441
442     /* The following function will read value of the register that contains the
443        gain of the digital filter*/
```

```
444
445 int Read_Filter_Gain(unsigned Motor_Address)
446 {
447     int Gain;
448
449     Gain = Read_from_motor_controller(Motor_Address, Filter_Gain_Address);
450     return Gain;
451 }
452
453
454
455
456
457 /* The following function will write data to the Max Acceleration register*/
458
459 void Write_Max_Accel(unsigned Motor_Address, int *Max_Acc_data)
460 {
461     int Acc_data = 0;
462
463     /* Output the most significant byte to the motor controller */
464     Acc_data = ((Max_Acc_data[0] >> 8) & 0xff) | (Max_Acc_data[1] & 0xff00);
465     Write_to_motor_controller(Motor_Address, Acc_data, Accel_MSB_Address);
466
467     /* Output the least significant byte to the motor controller */
468     Acc_data = (Max_Acc_data[0] & 0x00ff) | (Max_Acc_data[1] << 8);
469     Write_to_motor_controller(Motor_Address, Acc_data, Accel_LSB_Address);
470 }
471
472
473
474
475 /* The following function will read the Maximum acceleration register*/
476
477 int Read_Max_Accel(unsigned Motor_Address, int *Motor_Acceleration)
478 {
479     int Accel_data1, Accel_data2;
480
481     /* Read the most significant byte max. acceleration from the motor controller */
482     Accel_data2 = Read_from_motor_controller(Motor_Address, Accel_MSB_Address);
483
484     /* Read the least significant byte of the command position from the motor controller */
485     Accel_data1 = Read_from_motor_controller(Motor_Address, Accel_LSB_Address);
486
487     /* Shift and or the data read from motor controller and store in a 21 byte
488        word */
489
490     Motor_Acceleration[0] = Accel_data2 & 0xff;
491     Motor_Acceleration[0] = (Motor_Acceleration[0] << 8) | (Accel_data1 & 0xff);
492
493     Motor_Acceleration[1] = Accel_data2 & 0xff00;
494     Motor_Acceleration[1] = Motor_Acceleration[1] | ((Accel_data1 & 0xff00) >> 8);
495 }
496
497
498
499
500 /* The following function writes the Max Velocity for trapezoidal mode to the
501    to the motor controller */
502
503 void Write_Max_Velocity(unsigned Motor_Address, int MaxVelocity)
504 {
505     Write_to_motor_controller(Motor_Address, MaxVelocity, Max_Velocity_Address);
506 }
507
508
509
510
511 /* The following function reads the value of the register that contains the
512    Max velocity for trapezoidal mode*/
513
514 int Read_Max_Velocity(unsigned Motor_Address)
515 {
516     int Velocity;
517 }
```

```
518
519     Velocity = Read_from_motor_controller(Motor_Address,Max_Velocity_Address);
520     return Velocity;
521 }
522
523
524
525
526 /* The following function will write data to the Final position register*/
527
528 void Write_Final_pos(unsigned Motor_Address,long int *Final_data)
529 {
530     long int New_final_data[2];
531     int Fin_data = 0;
532
533     if (Motor_Address == Motor_Pair1_Address)
534     {
535         New_final_data[0] = Final_data[0] * -1;
536         New_final_data[1] = Final_data[1] * -1;
537     }
538     else
539     {
540         New_final_data[0] = Final_data[0];
541         New_final_data[1] = Final_data[1];
542     }
543
544
545     /* Output the most significant byte to the motor controller */
546     Fin_data = (((New_final_data[0] >> 16) & 0xff) | ((New_final_data[1] >> 8) & 0xff00));
547     Write_to_motor_controller(Motor_Address,Fin_data,Final_Pos3_Address);
548
549     /* Output the middle byte to the motor controller */
550     Fin_data = ((New_final_data[0] >> 8) & 0xff) | (New_final_data[1] & 0xff00);
551     Write_to_motor_controller(Motor_Address,Fin_data,Final_Pos2_Address);
552
553     /* Output the least significant byte to the motor controller */
554     Fin_data = ((New_final_data[0] & 0xff) | (New_final_data[1] << 8) & 0xff00);
555     Write_to_motor_controller(Motor_Address,Fin_data,Final_Pos1_Address);
556
557 }
558
559
560 /* The following function will read the Final motor position register*/
561
562 Read_Final_Pos(unsigned Motor_Address, long int *Final_pos)
563 {
564     int Final_data1,Final_data2,Final_data3;
565
566     /* Read the most significant byte of the command position from the motor controller */
567     Final_data3 = Read_from_motor_controller(Motor_Address,Final_Pos3_Address);
568
569     /* Read the middle byte of the command position from the motor controller */
570     Final_data2 = Read_from_motor_controller(Motor_Address,Final_Pos2_Address);
571
572     /* Read the least significant byte of the command position from the motor controller */
573     Final_data1 = Read_from_motor_controller(Motor_Address,Final_Pos1_Address);
574
575     /* Determine the final position of the drum motor */
576     Final_pos[0] = Final_data3 & 0xff;
577     Final_pos[0] = (Final_pos[0] << 8) | (Final_data2 & 0xff);
578     Final_pos[0] = (Final_pos[0] << 8) | (Final_data1 & 0xff);
579
580     /* Determine final position of the arm motor */
581     Final_pos[1] = Final_data3 & 0xff00;
582     Final_pos[1] = (Final_pos[1] << 8) | (Final_data2 & 0xff00);
583     Final_pos[1] = (Final_pos[1] | ((Final_data1 & 0xff00) >> 8));
584
585     if (Motor_Address == Motor_Pair1_Address)
586     {
587         Final_pos[0] = Final_pos[0] * -1;
588         Final_pos[1] = Final_pos[1] * -1;
589     }
590
591 }
```

```
592     if (Final_pos[0] > 0x7fffff)
593     Final_pos[0] -= 0xfffff;
594
595     if (Final_pos[1] > 0x7fffff)
596     Final_pos[1] -= 0xfffff;
597
598
599     }
600
```

```

1
2  /* Display routine is used to provide the menu for manual control of the motors */
3
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <graph.h>
9  #include <dos.h>
10 #include <time.h>
11 #include <stdlib.h>
12 #include <math.h>
13 #include <cport.h>
14
15 #define LEFTARROW 75
16 #define RIGHTARROW 77
17 #define UPARROW 72
18 #define DOWNARROW 80
19
20
21 signed char output_buffer[512];
22 signed char input_buffer[1024];
23 unsigned int rs;
24 int BLK_SZ = 8;
25
26 /* Declare the following variables external these var. are declared in motor.c */
27 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
28 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
29 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
30
31 extern unsigned Port_A_Address; /* Adress of port A on 8255 */
32 extern unsigned Port_B_Address; /* Adress of port B on 8255 */
33 extern unsigned Port_C_Address; /* Adress of port C on 8255 */
34 extern int Set_to_actual;
35 extern int C_Port_Data;
36
37 extern long int Starting_Position[2][3];
38 extern long int Current_Position[2][3];
39
40 extern int Stop_Serial_Data[3];
41 extern int Initiate_Serial_Tran[3];
42
43 extern struct Channel_data
44 {
45     unsigned Time_tag;
46     double Elevation_error;
47     double Azimuth_error;
48     double Signal_strength;
49     signed char Transmit_Data;
50 } Channel[3], Test[3];
51
52 long int timelapsed;
53
54 int Exercise_Multiple_Channels(int *Channels_to_Exercise)
55 {
56     int Switch_number_array[3][5] = {{3,4,5,6,3},{9,10,11,12,9},{15,16,17,18,15}};
57     long int Delta_move_array[3][4] = {{-500,40,0,500},{-500,40,0,500},{-500,40,0,500}};
58     long int Alpha_move_array[3][4] = {{0,100,-100,0},{0,100,-100,0},{0,100,-100,0}};
59     long int Alpha_move_only = {40};
60     long int Delta_move = {0};
61     int numb_switch, chan_number, Activated_switch, Address, Move_finished;
62     long int Target_Position[2];
63     long int Position_of_Motor[2];
64     int exit_loop = {0};
65
66     while (kbhit()) getch(); /* clear out keyboard buffer */
67
68     /* Start all requested channels running */
69     numb_switch = 4;
70
71     for (chan_number = 1; chan_number <=3; chan_number++)
72     {
73         if (Channels_to_Exercise[chan_number] == 1)

```



```
74     {
75         switch (chan_number)
76         {
77             case 1:
78                 { Address = Motor_Pair1_Address; break;}
79             case 2:
80                 { Address = Motor_Pair2_Address; break;}
81             case 3:
82                 { Address = Motor_Pair3_Address; break;}
83         }
84
85
86         Move_Motor(Address,Alpha_move_array[chan_number-1][numb_switch-1],
87                 Delta_move_array[chan_number-1][numb_switch-1]);
88
89         printf("\nMoving channel %i to limit switch %i",chan_number,
90                 Switch_number_array[chan_number-1][numb_switch]);
91
92     } /* End of if Channels_to_Exercise[chan_number] == 1 */
93 } /* End of chan_number <= 3 */
94
95 while (exit_loop != 1)
96 {
97     for (chan_number = 1; chan_number <=3; chan_number++)
98     {
99         if (kbhit()) exit_loop = 1;
100         while (kbhit()) getch(); /* clear out keyboard buffer */
101
102         if (Channels_to_Exercise[chan_number] == 1)
103         {
104             for (numb_switch = 1; numb_switch <= 4; numb_switch++)
105             {
106
107                 Activated_switch = Check_for_switch_Activation(Switch_number_array
108                         [chan_number-1][numb_switch-1]);
109
110                 if (Activated_switch == 1)
111                 {
112                     /* Delay program till motor stops */
113                     for (timelaped = 60000; timelaped > 0; timelaped = timelaped - 1);
114
115                     switch (chan_number)
116                     {
117                         case 1:
118                             { Address = Motor_Pair1_Address;
119                                 Alpha_move_only = 25;
120                                 break;
121                             }
122                         case 2:
123                             { Address = Motor_Pair2_Address;
124                                 Alpha_move_only = 40;
125                                 break;
126                             }
127                         case 3:
128                             { Address = Motor_Pair3_Address;
129                                 Alpha_move_only = 55;
130                                 break;
131                             }
132                     }
133
134                     if (numb_switch == 4)
135                     {
136                         /* Stop motors and set to actual position */
137                         Set_Motor_Position(Address,Set_to_actual);
138
139                         Read_Actual_Pos(Address,Position_of_Motor);
140
141                         Target_Position[0] = (Alpha_move_only * 769) + Position_of_Motor[0];
142                         Target_Position[1] = Position_of_Motor[1];
143
144                         Move_Motor(Address,Alpha_move_only,Delta_move);
145                         Move_finished = 0;
146
147                         /* Wait for arm to swing over before continuing */
```

```

148         while (Move_finished != 1)
149             Move_finished = Reached_Commanded_Pos(Address,Target_Position,chan_number,30);
150     } /* End of numb_switch == 4 */
151
152     Move_Motor(Address,Alpha_move_array[chan_number-1][numb_switch-1],
153             Delta_move_array[chan_number-1][numb_switch-1]);
154
155     printf("\nMoving channel %i to limit switch %i",chan_number,
156             Switch_number_array[chan_number-1][numb_switch]);
157
158     if (kbhit()) exit_loop = 1;
159     while (kbhit()) getch(); /* clear out keyboard buffer */
160
161     } /* End of if activated_switch == 1 */
162 } /* End of numb_switch <= 4 */
163     } /* End of Channels_to_Exercise[chan_number] == 1 */
164     } /* End of chan_number <=3 */
165 } /* End of exit_loop != 1*/
166
167 /* Stop motors and reset all positions counters to zero */
168 for (chan_number = 1; chan_number <=3; chan_number++)
169 {
170     switch (chan_number)
171     {
172         case 1: Address = Motor_Pair1_Address;
173         case 2: Address = Motor_Pair2_Address;
174         case 3: Address = Motor_Pair3_Address;
175     }
176     Set_Motor_Position(Address,Set_to_actual);
177 }
178
179 } /* End of function */
180
181 int Move_Motor(unsigned Motor_Address,long int Move_to_Alpha, long int Move_to_Delta)
182 {
183     {
184         long int Actual_Pos[2];
185         long int New_Pos[2];
186
187         /* Stop motors and set to actual position */
188         Set_Motor_Position(Motor_Address,Set_to_actual);
189
190         /* Delay reading of motor till it as a chance to move */
191         for (timelaped = 50000; timelaped > 0; timelaped = timelaped - 1);
192
193         Read_Actual_Pos(Motor_Address,Actual_Pos);
194
195         /*Set up positions for the move command */
196         /* Add delta to Disk B and Disk A motor position */
197         New_Pos[0] = (Move_to_Alpha * 769) + (1967 * Move_to_Delta) + Actual_Pos[0];
198         New_Pos[1] = Actual_Pos[1] - (1967 * Move_to_Delta);
199
200         /* Output new position to the motor controller */
201         Write_Final_pos(Motor_Address,New_Pos);
202
203         /* Start trapezoidal move */
204         Write_to_flag_register (Motor_Address,0x0808);
205
206         /* Delay continuing till of motor till it as a chance to move */
207         for (timelaped = 100000; timelaped > 0; timelaped = timelaped - 1);
208
209     }
210
211 void Position(double Azimuth, double Elevation, int channel,
212             unsigned Motor_Address,int Incremental)
213 {
214     {
215         double Delta[4], Alpha[4],AZ_Deg,EL_Deg,Command_Length,U_Scale,V_Scale;
216         double Level_of_Signal,Level_of_UError,Level_of_VError;
217         long int I_StepA[4], I_StepB[4], Step_Dif;
218         long int Command_Pos[2],Actual_Pos[2];
219         int a,b, Number_of_Solutions,loop,Solution;
220         long int Present_Delta = 10000000;

```

```
222     int t = 1;
223     long int time_out;
224     int Numb_of_Channels, Loop_count;
225     Loop_count = 0;
226     Level_of_Signal = 0;
227     Level_of_UError = 0;
228     Level_of_VError = 0;
229
230     AZ_Deg = (Azimuth) * 57.29577951;
231     EL_Deg = (Elevation) * 57.29577951;
232
233     AzEl2AB(Azimuth, Elevation, I_StepA, I_StepB, Delta, Alpha, &Number_of_Solutions);
234
235     for (loop=0; loop <= Number_of_Solutions; loop++)
236     {
237         Step_Dif = Current_Position[1][channel-1] - I_StepA[loop];
238         if (labs(Step_Dif) < Present_Delta)
239         {
240             Present_Delta = labs(Step_Dif);
241             Solution = loop;
242         }
243     }
244
245     _clearscreen( _GCLEARSCREEN );
246     printf("Use cursor keys to move the head in the X and Y coordinates.\n");
247     printf("Press 'E' to exit.\n");
248
249     Read_Actual_Pos(Motor_Address, Actual_Pos);
250
251     Actual_Pos[0] = Starting_Position[0][channel-1] - Actual_Pos[0];
252     Actual_Pos[1] = Starting_Position[1][channel-1] + Actual_Pos[1];
253
254     if (channel == 1) Numb_of_Channels = 2;
255     else if (channel == 2) Numb_of_Channels = 1;
256     else Numb_of_Channels = 3;
257
258     if (channel == 3)
259     {
260         U_Scale = - 0.0002718;
261         V_Scale = 0.0002453;
262     }
263     else
264     {
265         U_Scale = 0.0002718;
266         V_Scale = - 0.0002453;
267     }
268
269     do
270     {
271
272         _settextposition(4,5); /*Position the cursor at position 1,5)*/
273
274         AZ_Deg = (Azimuth) * 57.29577951;
275         EL_Deg = (Elevation) * 57.29577951;
276
277         printf("Channel %i Position\n", Numb_of_Channels);
278         printf("\n\tAzimuth: %8.6f\t\t Elevation: %8.6f\n",
279             AZ_Deg, EL_Deg);
280         printf("\n\tDelta: %10.6f\t\t Alpha: %10.6f\n",
281             Delta[Solution], Alpha[Solution]);
282         printf("\n\tI_StepA: %li\t\t I_StepB: %li\n",
283             I_StepA[Solution], I_StepB[Solution]);
284         printf("\n\tActualA: %li\t\t ActualB: %li\n",
285             Actual_Pos[1], Actual_Pos[0]);
286
287     } do {
288
289         /* Turn DSP for data retrieval */
290         C_Port_Data = C_Port_Data | Initiate_Serial_Tran[channel - 1];
291         outp(Port_C_Address, C_Port_Data);
292
293         ComFlushRx(); /* Flush out data before starting loop */
294
295         time_out = 0;
```

```
296
297     /* Wait for data before reading out data */
298     while ((ComLenRx() < BLK_SZ) && (time_out <= 50000))
299         time_out = time_out + 1;
300
301     if (time_out < 50000)
302     {
303         Read_Channel_Data(&Test[channel - 1]);
304     }
305     /* Clear out keyboard buffer before continuing */
306     else
307     {
308         while (kbhit()) b = getch();
309         ComFlushRx();
310     }
311
312     /* Turn off sync pulse to x channel */
313     C_Port_Data = C_Port_Data & Stop_Serial_Data[channel - 1];
314     outp(Port_C_Address, C_Port_Data);
315
316     Level_of_Signal = Test[channel-1].Signal_strength/204.7 + Level_of_Signal;
317     Level_of_UError = Test[channel-1].Azimuth_error*U_Scale + Level_of_UError;
318     Level_of_VError = Test[channel-1].Elevation_error*V_Scale + Level_of_VError;
319     Loop_count++;
320
321     if (Loop_count == 10)
322     {
323         _settextposition(16,1); /*Position the cursor at position 1,5)*/
324
325         printf("\n\tTime_tag = %u\t Signal Level: %10.6f\n",
326             Test[channel-1].Time_tag,
327             Level_of_Signal/Loop_count);
328
329         printf("\n\tU_Error: %10.6f \t V_Error: %10.6f \n",
330             Level_of_UError/Loop_count,
331             Level_of_VError/Loop_count);
332
333         Level_of_Signal = 0;
334         Level_of_UError = 0;
335         Level_of_VError = 0;
336         Loop_count = 0;
337     }
338
339     ) while (!kbhit());
340
341     a=getch();
342
343     if (a==0) a=getch();
344
345     switch(a)
346     { case LEFTARROW:    Azimuth+=10e-6;
347                         break;
348
349
350
351     case RIGHTARROW:    Azimuth-=10e-6;
352                         break;
353
354     case UPARROW:       Elevation-=10e-6;
355                         break;
356
357     case DOWNARROW:     Elevation+=10e-6;
358                         break;
359
360     case '4':           Azimuth+=100e-6;
361                         break;
362
363     case '6':           Azimuth-=100e-6;
364                         break;
365
366     case '2':           Elevation+=100e-6;
367                         break;
368
369     case '8':           Elevation-=100e-6;
```

```
370             break;
371
372     default: break;
373     }
374
375     Read_Actual_Pos(Motor_Address,Actual_Pos);
376
377     Actual_Pos[0] = Starting_Position[0][channel-1] - Actual_Pos[0];
378     Actual_Pos[1] = Starting_Position[1][channel-1] + Actual_Pos[1];
379
380
381     AzEl2AB(Azimuth, Elevation, I_StepA, I_StepB, Delta,
382     Alpha,&Number_of_Solutions);
383
384     /* Search for the nearest solution */
385     Present_Delta = 10000000;
386
387     for (loop=0; loop <= Number_of_Solutions; loop++)
388     {
389         Step_Dif = Actual_Pos[1] - I_StepA[loop];
390         if (labs(Step_Dif) < Present_Delta)
391         {
392             Present_Delta = labs(Step_Dif);
393             Solution = loop;
394         }
395     }
396
397     /* Set up the commanded position for the DiskB Motor */
398     Command_Pos[0] = Starting_Position[0][channel-1] - I_StepB[Solution];
399
400     /* Set up the commanded position for the DiskA Motor */
401     Command_Pos[1] = (-1 * Starting_Position[1][channel-1]) + I_StepA[Solution];
402
403     /* Output new position to the motor controller */
404     Write_Final_pos(Motor_Address,Command_Pos);
405
406     /* Start trapezoidal move */
407     Write_to_flag_register (Motor_Address,0x0808);
408
409     while (kbhit()) b = getch();
410
411     } while ((a!='e') && (a!='E'));
412
413 }
414
415
416
417 void Run_Scan_Pattern(double Azimuth, double Elevation, int channel,
418     unsigned Motor_Address,double trip_point)
419
420 { double Delta[4], Alpha[4],AZ_Deg,EL_Deg,Command_Length;
421     long int I_StepA[4], I_StepB[4], Step_Dif;
422     long int Command_Pos[2],Actual_Pos[2];
423     int key,b, Number_of_Solutions,loop,Solution,change_par;
424     long int Present_Delta = 10000000;
425     int t = 1;
426     long int time_out,loop_count,setpoint;
427     int exit = 0;
428
429     AZ_Deg = (Azimuth) * 57.29577951;
430     EL_Deg = (Elevation) * 57.29577951;
431
432     _clearscreen( _GCLEARSCREEN );
433
434     AzEl2AB(Azimuth, Elevation, I_StepA, I_StepB, Delta, Alpha, &Number_of_Solutions);
435
436     for (loop=0; loop <= Number_of_Solutions; loop++)
437     {
438         Step_Dif = Current_Position[1][channel-1] - I_StepA[loop];
439         if (labs(Step_Dif) < Present_Delta)
440         {
441             Present_Delta = labs(Step_Dif);
442             Solution = loop;
443         }
444     }
```

```
444     }
445
446     Read_Actual_Pos(Motor_Address,Actual_Pos);
447
448     Actual_Pos[0] = Starting_Position[0][channel-1] - Actual_Pos[0];
449     Actual_Pos[1] = Starting_Position[1][channel-1] + Actual_Pos[1];
450
451     setpoint = 0;
452     Test[channel-1].Signal_strength = 0.0;
453
454     while ((Test[channel-1].Signal_strength/204.7 < trip_point) && (exit == 0))
455     for (change_par = 1; change_par <= 4; change_par++)
456     {
457         /* Exit if signal level is above trip point */
458         if (((Test[channel-1].Signal_strength /204.7) > trip_point) || (exit == 1))break;
459
460         if ((change_par == 1) || (change_par == 3))
461             setpoint = setpoint + 1;
462
463         for (loop_count = 1; loop_count <= setpoint; loop_count++)
464         {
465             if (kbhit())
466             { key = getch();
467               if ((key == 'e') || (key == 'E')) exit = 1;
468             }
469
470             _settextposition(2,5); /*Position the cursor at position 1,5)*/
471
472             AZ_Deg = (Azimuth) * 57.29577951;
473             EL_Deg = (Elevation) * 57.29577951;
474
475             printf("Position:\n\tAzimuth: %8.6f      Elevation: %8.6f\n",
476                  AZ_Deg, EL_Deg);
477             printf("\n\tDelta: %10.6f      Alpha: %10.6f\n",
478                  Delta[Solution],Alpha[Solution]);
479             printf("\n\tI_StepA: %li      I_StepB: %li\n",
480                  I_StepA[Solution],I_StepB[Solution]);
481             printf("\n\tActualA: %li      ActualB: %li\n",
482                  Actual_Pos[1],Actual_Pos[0]);
483             printf("\n\tSignal strength = %3.6f",
484                  Test[channel-1].Signal_strength/204.7 );
485
486             /* Turn DSP for data retrieval */
487             C_Port_Data = C_Port_Data | Initiate_Serial_Tran[channel - 1];
488             outp(Port_C_Address,C_Port_Data);
489
490             ComFlushRx(); /* Flush out data before starting loop */
491
492             time_out = 0;
493
494             /* Wait for data before reading out data */
495             while ((ComLenRx() < BLK_SZ) && (time_out <= 80000))
496                 time_out = time_out + 1;
497
498             if (time_out < 80000)
499                 Read_Channel_Data(&Test[channel - 1]);
500             else if (time_out >= 80000) ComFlushRx();
501
502             /* Turn off sync pulse to x channel */
503             C_Port_Data = C_Port_Data & Stop_Serial_Data[channel - 1];
504             outp(Port_C_Address,C_Port_Data);
505
506             /* Break out of for loop if signal is large enough */
507             if (((Test[channel-1].Signal_strength /204.7) > trip_point) || (exit == 1)) break;
508
509             switch(change_par)
510             { case 1:      Azimuth+=50e-6;
511                           break;
512               case 2:      Elevation+=50e-6;
513                           break;
514               case 3:      Azimuth-=50e-6;
515                           break;
516             }
```

```
518     case 4:          Elevation-=50e-6;
519                     break;
520     default: break;
521 }
522
523 Read_Actual_Pos(Motor_Address,Actual_Pos);
524
525 Actual_Pos[0] = Starting_Position[0][channel-1] - Actual_Pos[0];
526 Actual_Pos[1] = Starting_Position[1][channel-1] + Actual_Pos[1];
527
528 AzEl2AB(Azimuth, Elevation, I_StepA, I_StepB, Delta,
529         Alpha,&Number_of_Solutions);
530
531     /* Search for the nearest solution */
532     Present_Delta = 10000000;
533
534     for (loop=0; loop <= Number_of_Solutions; loop++)
535     {
536         Step_Dif = Actual_Pos[1] - I_StepA[loop];
537         if (labs(Step_Dif) < Present_Delta)
538         {
539             Present_Delta = labs(Step_Dif);
540             Solution = loop;
541         }
542     }
543
544     /* Set up the commanded position for the DiskB Motor */
545     Command_Pos[0] = Starting_Position[0][channel-1] - I_StepB[Solution];
546
547     /* Set up the commanded position for the DiskA Motor */
548     Command_Pos[1] = (-1 * Starting_Position[1][channel-1]) + I_StepA[Solution];
549
550     /* Output new position to the motor controller */
551     Write_Final_pos(Motor_Address,Command_Pos);
552
553     /* Start trapezoidal move */
554     Write_to_flag_registor (Motor_Address,0x0808);
555
556     } /* End of for (loop_conut = 1; loop_count <= setpoint; loop_count++) */
557
558 } /* End o for (change_par = 1; change_par < 4; change_par++) */
559
560     /* Turn off sync pulse to x channel */
561     C_Port_Data = C_Port_Data & Stop_Serial_Data[channel - 1];
562     outp(Port_C_Address,C_Port_Data);
563
564 }
565
566
```

```
1  /*
2   The following include statements are required to allow
3   program to link up with Turbo C++ Libraries */
4
5  #include <cport.h>
6  #include <bios.h>
7  #include <conio.h>
8  #include <dos.h>
9  #include <ctype.h>
10 #include <stdio.h>
11 #include <math.h>
12 #define BLK_SZ 8
13
14 signed char output_buffer[512];
15 signed char input_buffer[1024];
16 unsigned int rs;
17 int V_error,U_error>Total_signal_strength;
18 int Frame_count,Number_of_samples,Toggle_count,Process_data_flag,Time_tag;
19 char Mode_flag;
20 char tmpbuf[128]; /* Char array for storing time data */
21 char tmpbuf1[128]; /* Char array for storing time data */
22
23 struct Channel_data
24 {
25     unsigned Time_tag;
26     double V_error;
27     double U_error;
28     double Signal_strength;
29     signed char Transmit_Data;
30 } Channel[3],Test[3];
31
32 double Offset = .03;
33
34 extern unsigned Port_A_Address; /* Address of port A on 8255 */
35 extern unsigned Port_B_Address; /* Address of port B on 8255 */
36 extern unsigned Port_C_Address; /* Address of port C on 8255 */
37 extern FILE *output_ptr; /* Pointer for file to output error data */
38
39 void Init_Com_port(void);
40 void Read_Channel_Data(struct Channel_data *Ch_ptr);
41 int Test_Serial_Link(int Chan_Number,int Port_C_Data);
42
43 /* Read_Channel_Data reads in the data and formats as required */
44 void Read_Channel_Data(struct Channel_data *Ch_ptr)
45 {
46     /* Read data from dsp via com port*/
47     rs = ComIn(input_buffer,BLK_SZ);
48
49     /* Send over data to the monitor PC */
50     ComActive(COM2);
51     ComOut(&(Ch_ptr->Transmit_Data),1);
52     ComOut(input_buffer,rs);
53     ComActive(COM3);
54
55     /*Convert input data to 16 bit format for time tag
56     input_buffer[0] and input_buffer[1] time tag data*/
57
58     Ch_ptr->Time_tag = (int)input_buffer[1];
59     Ch_ptr->Time_tag = (Ch_ptr->Time_tag & 0xff) << 8;
60     Ch_ptr->Time_tag += (int)input_buffer[0] & 0xff;
61
62     /* Convert buffer inputs 2,3 to 16 bit elevation error */
63     V_error = (int)input_buffer[3];
64     V_error = (V_error & 0xff) << 8;
65     V_error += (int)input_buffer[2] & 0xff;
66
67     /* Convert buffer inputs 4,5 to 16 bit azimuth error */
68     U_error = (int)input_buffer[5];
69     U_error = (U_error & 0xff) << 8;
70     U_error += (int)input_buffer[4] & 0xff;
71
72     /* Convert buffer inputs 6,7 to 16 bit total_strength error */
73
```



```
74     Total_signal_strength = (int)input_buffer[7];
75     Total_signal_strength = (Total_signal_strength & 0xff) << 8;
76     Total_signal_strength += (int)input_buffer[6] & 0xff;
77
78     /*Change integer value to a floating point value for normalizing*/
79     Ch_ptr->Signal_strength = (double)Total_signal_strength - Offset;
80
81     if (Ch_ptr->Signal_strength < 0.00006)
82     {
83         /* Set signal_strength to small number if equal 0.0 */
84         Ch_ptr->Signal_strength = 0.00006;
85     }
86
87     /* Normalize azimuth error */
88     Ch_ptr->U_error = (double)U_error / Ch_ptr->Signal_strength;
89
90     /* Normalize Elevation error */
91     Ch_ptr->V_error = (double)V_error / Ch_ptr->Signal_strength;
92
93     } /* End of read channel data */
94
95
96
97 /* Init_Com_port initializes serial ports */
98
99 void Init_Com_port(void)
100 {
101     int rv = 0;
102     /* Initialize RS-232 port Com2 will be for sending data a monitoring
103     PC and Com3 will be ty\he receiver for DSP Data*/
104     /* rv = 2 bad 'com' parameter, rv = 3 no uart chip detected */
105     /* rv = 4 receive queue allocation error, rv = 4 transmit queue allocation error */
106
107     /* the following line will be used for the compuadd computer at 115kbaud */
108     rv = ComOpen(COM3, B115200, W8|S1|NONE, 16, 16);
109
110     /* Used for 19.2Kb option */
111     rv = ComOpen(COM2, B19200, W8|S1|NONE, 16, 16);
112
113     /* Set up COM2 port to be active */
114     ComActive(COM2);
115
116     /* Remove all data from the receive buffer */
117     ComFlushRx();
118
119     /* Remove all data from the transmit buffer */
120     ComFlushTx();
121
122     /* Set up COM3 port to be active */
123     ComActive(COM3);
124
125     /* Remove all data from the receive buffer */
126     ComFlushRx();
127
128     /* Remove all data from the transmit buffer */
129     ComFlushTx();
130
131     } /* End of Init_Com_port */
132
133
134 /* Function Test_Serial_Link is used to test serial link from DSP
135 to PC */
136
137 int Test_Serial_Link(int Chan_Number,int Port_C_Data)
138 {
139     int Stop_Serial_Data[3] = {0xBF,0x7F,0xEF};
140     int Initiate_Serial_Tran[3] = {0x40,0x80,0x10};
141     int Data_Link_Length = 0;
142     int Serial_Link_Error = 0;
143     int Time_out = 0;
144     int long_test_loop;
145     int delay_loop;
146     int Error_number; /* Initialize Error number to be sent to
147                        calling program */
```

```
148     Time_out = 0;
149     Error_number = 0; /*Initialize error number to zero */
150
151     Port_C_Data = Port_C_Data | Initiate_Serial_Tran[Chan_Number - 1];
152     outp(Port_C_Address,Port_C_Data);
153
154     for (test_loop = 1; test_loop <= 50; test_loop++)
155     {
156         Time_out = 0;
157
158         /* Check to see if data is available from the receiver electronics */
159
160         while (ComLenRx() < BLK_SZ)
161         {
162             Data_Link_Length = ComLenRx();
163             Serial_Link_Error = ComError();
164
165             if (Serial_Link_Error != 0)
166             {
167                 /* Display date and time of error*/
168                 printf("\nRS232 link error detected %x\n",Serial_Link_Error);
169                 printf("RS232 message length %i\n",Data_Link_Length);
170                 Error_number = 1;
171                 break;
172             }
173
174             Time_out++; /* Increment timeout count for serial port */
175
176             if (Time_out == 32000)
177             {
178                 printf ("\nSerial link timeout error\n ");
179                 Error_number = 2;
180                 break;
181             }
182         } /* End of while (ComLenRx() < BLK_SZ) */
183
184         Data_Link_Length = ComLenRx();
185         /* If data length is greater than report error */
186         if (Data_Link_Length == 8) Read_Channel_Data(&Test[Chan_Number - 1]);
187         else
188         {
189             printf("\nMessage length error. Length of messege = %i\n",Data_Link_Length);
190             ComFlushRx();
191         }
192
193         if (Error_number != 0) break; /* Exit for loop if failure has occurred */
194
195         for (delay_loop = 1; delay_loop <= 5000; delay_loop++);
196         ComFlushRx();
197     } /* End of for test loop */
198
199     /* Turn off sync pulse to x channel */
200     Port_C_Data = Port_C_Data & Stop_Serial_Data[Chan_Number - 1];
201     outp(Port_C_Address,Port_C_Data);
202
203     return Error_number;
204 }
205
206 void Report_Com_Error(int Error_on_link,int Channel)
207 {
208     int Data_length;
209     /* Display date and time of error*/
210     Data_length = ComLenRx();
211     _strtime( tmpbuf);
212     _strdate( tmpbuf1);
213     fprintf(output_ptr, "\n Time & Date of error = %s    %s ",
214             tmpbuf,tmpbuf1);
215     fprintf(output_ptr,"\nRS232 link error detected %x on Channel %i \n",
216             Error_on_link,Channel);
217     fprintf(output_ptr,"RS232 message length %i\n",Data_length);
218 }
219
220
221
```

```
222  /* Function Serial_Port_Timeout reports timeout failure of serial port */
223
224  void Serial_Port_Timeout(int Channel)
225  {
226      printf("\nSerial Port timeout has occurred on Channel %i", Channel);
227      _strtime( tmpbuf );      /* Retrieve system time and date */
228      _strdate( tmpbuf1);
229      printf("\n Time of timeout = %s", tmpbuf);
230      fprintf(output_ptr, "\n Time & Date of error = %s      %s ",
231              tmpbuf,tmpbuf1);
232      fprintf(output_ptr,"\nSerial Port timeout has occurred on channel %i",Channel);
233
234  }
235
236  void Initialize_Dsp_Link(void)
237  {
238      int Stop_Data[3] = {0xBF,0x7F,0xEF};
239      int Initiate_Serial[3] = {0x40,0x80,0x10};
240      long int long_delay;
241      int Init_Chan;
242      int Port_data = {0x00};
243
244      for (Init_Chan = 1; Init_Chan <= 3; Init_Chan++)
245      {
246          /* Turn on sync pulse to DSP */
247          Port_data = Port_data | Initiate_Serial[Init_Chan];
248          outp(Port_C_Address,Port_data);
249
250          for (long_delay = 1; long_delay <= 500000; long_delay++);
251
252          /* Turn off sync pulse to x channel */
253          Port_data = Port_data & Stop_Data[Init_Chan];
254          outp(Port_C_Address,Port_data);
255      }
256
257  } /* End of function Initialize_Dsp_Link */
```

```

1
2  /* Track.C: Main executive for controlling the tracking software */
3
4  #include <stdio.h>
5  #include <conio.h>
6  #include <ctype.h>
7  #include <time.h>
8  #include <dos.h>
9  #include <time.h>
10 #include <cport.h>
11 #include <math.h>
12 #include <stdlib.h>
13 #include <graph.h>
14
15 void Move_Chan_To_Starting_Pos(void);
16 void Init_Channel_Address(void);
17 void Compute_Offset(int I_chan);
18 void Init_Array(void);
19
20 extern void _fortran POSITION(int _far *ICHANNEL_mod, int long _far *INTRACK,
21                             double _far *U, double _far *V,
22                             long int _far *ISTEPA, long int _far *ISTEPB);
23
24 extern void _fortran INIT(int long _far *REALTIME);
25
26 extern unsigned Control_word_address; /* Address to which 8255 command word
27                                     is being set */
28 extern unsigned Control_word; /* Sets all ports A,B to inputs &
29                               C to outputs */
30
31 /* Declare the following variables external these var. are declared in motor.c */
32 extern unsigned Motor_Pair1_Address; /*Address to first pair of motor controllers */
33 extern unsigned Motor_Pair2_Address; /*Address to first pair of motor controllers */
34 extern unsigned Motor_Pair3_Address; /*Address to first pair of motor controllers */
35
36 /* Declare all variables used for 8255 control */
37 extern unsigned Control_word_address; /* Address of Control word on 8255 */
38 extern unsigned Port_A_Address; /* Address of port A on 8255 */
39 extern unsigned Port_B_Address; /* Address of port B on 8255 */
40 extern unsigned Port_C_Address; /* Address of port C on 8255 */
41
42 extern int Reset_to_zero; /* These 2 variables will be used global for setting motor position */
43 extern int Set_to_actual;
44
45 /* Array of data used to read in RS232 data back from the DSP */
46 extern struct Channel_data
47 {
48     unsigned Time_tag;
49     double V_error;
50     double U_error;
51     double Signal_strength;
52     signed char Transmit_Data;
53 } Channel[3], Test[3];
54
55 unsigned Channel_Time_Tag[4];
56 int Acquire_New_Time_Tag[4];
57
58 /* Declarations for channel bias offsets */
59 double Uoffset[4] = {0.0, 0.0, -0.5, -0.446};
60 double Voffset[4] = {0.0, 0.0, -0.2, -0.390};
61 double U_Delta[6] = {0.00, 1.0, -2.0, 1.0, 0.00, 0.00};
62 double V_Delta[6] = {0.00, 0.00, 0.00, 1.0, -2.0, 0.00};
63 double MaxU_Delta[6] = {0.00, 1.00, -1.00, 0.00, 0.00, 0.00};
64 double MaxV_Delta[6] = {1.00, 1.00, 1.00, 2.00, 0.00, 0.00};
65 double Step_size = .05;
66
67 int Sample_Counts = 100;
68 int Sample_Count[4] = {0, 0, 0, 0};
69 int Inter_Count[4] = {0, 0, 0, 0};
70 double Signal_Pwr[4][6] = {{0.0, 0.0, 0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
71                             {0.0, 0.0, 0.0, 0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0, 0.0, 0.0}};
72
73 /* Threshold for determining intrack condition */

```

```
74 double track_threshold[4] = {0.5,0.5,0.5,0.5};
75 double High_threshold = 0.5;
76 double Low_threshold = 0.1;
77
78 long int Offset_Position[3][4];
79 unsigned int Channel_Address[4];
80
81 extern FILE *output_ptr; /* Pointer for file to output error data */
82 extern FILE *Data_Ptr; /* Pointer for file to output test data */
83 extern FILE *System_Ptr; /* Pointer for file to read and store system parameters */
84 extern FILE *AzEl_Ptr;
85 extern FILE *Align_Ptr;
86 extern FILE *Track_AzEl_Ptr;
87 extern FILE *Input_Ptr;
88
89 extern long int Starting_Position[2][3];
90 extern int Selected_Channels[3][2];
91
92 int Port_Data = 0x00; /* Initialize variable for port c data reg. */
93 int Pwr_24Volts_On = 0x20; /* Variable used to turn on 24 volt supply */
94 int Pwr_24Volts_Off = 0xDF; /* Variable used to turn off 24 volt supply */
95
96 int Track_Option; /* Flag used to determine what tracking option to do */
97 long int In_track = 0; /* Flag passed to control software indicating intrack position */
98 long int Track_Flag[4] = {0,0,0,0}; /* Array used to used to store intrack flag */
99
100 double U_signal = 0.0; /* Passes U error to control software */
101 double V_signal = 0.0; /* Passes V error to control software */
102 long int I_stepA = 0; /* Control software commanded position for arm */
103 long int I_stepB = 0; /* Control software commanded position for drum */
104 long int Act_pos[3];
105 long int Present_Position[3];
106
107 int long Real_time = 1; /* Flag used to indicate real_time mode */
108 long int Motor_Command[3];
109 int BLK_SZ = 8; /* Size of message from DSP in bytes */
110 int First_time_Flag = 1;
111 int Closed_Loop_Test = 1; /* Set to zero for open loop testing */
112 int first_time_in_track[4] = {0,0,0,0};
113 int Out_of_track_count[4] = {0,0,0,0};
114
115 int Loop_count = 1;
116 int count, loop;
117 int Number_of_Samples = 0;
118
119 extern int Channel_to_Run[4];
120 extern int Number_of_Channels_To_Run;
121
122 int Track_Channel = 1;
123 int I_channel = 2;
124 int Active_Channel;
125
126 /* Array used to disable sync pulse generation */
127 int Disable_Sync_Pulse[4] = {0x00,0xBF,0x7F,0xEF};
128
129 /* Array used enable sync pulse generation */
130 int Enable_Sync_Pulse[4] = {0x00,0x40,0x80,0x10};
131
132 char tmpbuf[128]; /* Char array for storing time data */
133 char tmpbuf1[128]; /* Char array for storing time data */
134 long int delay;
135
136 void Compute_Open_Loop_Commands(void); /* Set up function prototype */
137 void Wait_for_key_Press(void);
138 void Check_for_Key_Pressed(void);
139 void Sync_Serial_Link(void);
140
141 unsigned Link_error = {0};
142 unsigned Data_length = {0};
143 unsigned Time_out_count = {0};
144 int Record_Flag[4] = {0,0,0,0}; /* Set recording data option to zero */
145 char key_buffer[80];
146 int Align = 0;
147 int mode = 1;
```

```
148
149 void main ()
150 {
151
152     /* Assign channel assignments to data array */
153     Channel[0].Transmit_Data=0;
154     Channel[1].Transmit_Data=1;
155     Channel[2].Transmit_Data=2;
156
157     Screen_Prompt();
158     _clearscreen( _GCLEARSCREEN );
159
160     Open_files(mode); /* Open all required files mode = 1 Indicates track mode */
161
162     /* Output control word to 8255 */
163     Port_Data = 0x00;
164     outp(Control_word_address, Control_word);
165     outp(Port_C_Address,Port_Data);
166
167     Init_Array();
168
169     Select_Channels_to_Run(); /* Request the user what channels to run */
170
171     Select_Azel_Changes(); /*Request user for new azimuth and elevation parameters*/
172
173     /* Close files so that Avtec software can read data */
174     fclose(Input_Ptr);
175     fclose(Track_AzEl_Ptr);
176
177     INIT(&Real_time); /* Initialize Avtec control software */
178
179     Init_Channel_Address(); /* Address of motor controller assignment*/
180
181     /* Turn off 24 Volt power supply */
182     Port_Data = Port_Data & Pwr_24Volts_Off;
183     outp(Port_C_Address,Port_Data);
184
185     Initialize_Controllers(); /* Initialize controllers to a predefined parameters */
186
187     /* Turn on 24 Volt power supply */
188     Port_Data = Port_Data | Pwr_24Volts_On;
189     outp(Port_C_Address,Port_Data);
190
191     for (delay = 1; delay <= 70000; delay++);
192
193     Check_Switch_Status();
194     Check_System_Power();
195
196     for (loop = 1; loop <= Number_of_Channels_To_Run; loop ++ )
197     {
198         track_threshold[loop] = High_threshold;
199         I_channel = Channel_to_Run[loop];
200         Offset_Position[0][I_channel] = Starting_Position[0][I_channel-1];
201         Offset_Position[1][I_channel] = -Starting_Position[1][I_channel-1];
202     }
203     Move_Chan_To_Starting_Pos();
204
205
206     /* Initialize serial port for receiving data */
207     Init_Com_port(); /* Initialize com port for receiving data*/
208
209     _clearscreen( _GCLEARSCREEN );
210
211     printf("\n\n Tracking program is now running !!");
212     printf("\n Press 'e' to exit program");
213     printf("\n\n      ** Channel Status ** ");
214
215
216     for (loop = 1; loop <= Number_of_Channels_To_Run; loop ++ )
217     {
218         I_channel = Channel_to_Run[loop];
219         Port_Data = Port_Data | Enable_Sync_Pulse[I_channel];
220     }
221
```

```
222     outp(Port_C_Address,Port_Data); /* Send out command to initiate sync pulses */
223
224     /* Sync host pc to receiver electronics */
225     Sync_Serial_Link();
226
227     while(1)
228     {
229         for (Track_Channel = 1; Track_Channel <= Number_of_Channels_To_Run;
230             Track_Channel++)
231         {
232             I_channel = Channel_to_Run[Track_Channel];
233
234             /* If only one channels is running make sure that sync pulse goes
235             high before continuing */
236
237             if (Number_of_Channels_To_Run == 1)
238                 while (!I_channel == Check_for_Active_Sync_Pulse());
239
240             /* Wait till sync pulse goes low */
241             while (I_channel != Check_for_Active_Sync_Pulse());
242
243             Active_Channel = Check_for_Active_Sync_Pulse();
244             /* Check to see if data is available from the receiver electronics
245             Sync pulse is still active*/
246
247             while ((ComLenRx() < BLK_SZ) &&
248                 (I_channel == Active_Channel))
249             {
250                 Link_error = ComError();
251
252                 if (Link_error != 0)
253                     Report_Com_Error(Link_error,I_channel);
254
255                 Active_Channel = Check_for_Active_Sync_Pulse();
256             } /* End of while (ComLenRx() < BLK_SZ) */
257
258             Port_Data = Port_Data | 0x01;
259             outp(Port_C_Address,Port_Data); /* Send out command to initiate sync pulses */
260
261             /* Read data from serial buffer and format data as required */
262             if (ComLenRx() >= BLK_SZ)
263             { Read_Channel_Data(&Channel[I_channel - 1]);
264               if (Aquire_New_Time_Tag[I_channel] == 1)
265               {
266                   Channel_Time_Tag[I_channel] = Channel[I_channel - 1].Time_tag;
267                   Aquire_New_Time_Tag[I_channel] = 0;
268               }
269             }
270             else
271             {
272                 Channel[I_channel - 1].Signal_strength = 1.0;
273                 ComFlushRx();
274             }
275
276
277             if (Channel[I_channel - 1].Time_tag == Channel_Time_Tag[I_channel])
278             { /* Increment next time tag for next frame comparison */
279                 Channel_Time_Tag[I_channel]++;
280
281             }
282
283             /* Test to see if errors are out of range */
284             if (((Channel[I_channel - 1].Signal_strength/204.7) >= track_threshold[I_channel])
285                 && (fabsl(Channel[I_channel - 1].V_error) < 1.0) &&
286                 (fabsl(Channel[I_channel - 1].U_error) < 1.0))
287             {
288                 Track_Flag[I_channel] = 1;
289                 Sample_Count[I_channel]++;
290
291                 if (Sample_Count[I_channel] > 5)
292                     Signal_Pwr[I_channel][Inter_Count[I_channel]] =
293                     Channel[I_channel - 1].Signal_strength +
294                     Signal_Pwr[I_channel][Inter_Count[I_channel]];
295             }
```

```
296     if (Sample_Count[I_channel] == Sample_Counts)
297     {
298         Compute_Offset(I_channel);
299         Sample_Count[I_channel] = 0;
300     }
301
302
303     /* Convert normalized values into microradians */
304     /* Channel 1 & 2 have a sign reversal */
305     if (I_channel == 3)
306     {
307         U_signal = (Channel[I_channel - 1].U_error + Uoffset[I_channel])
308             * -0.0002718;
309         V_signal = (Channel[I_channel - 1].V_error + Voffset[I_channel])
310             * 0.0002453;
311     }
312     else
313     {
314         U_signal = (Channel[I_channel - 1].U_error + Uoffset[I_channel])
315             * 0.0002718;
316         V_signal = (Channel[I_channel - 1].V_error + Voffset[I_channel])
317             * -0.0002453;
318     }
319
320     if (first_time_in_track[I_channel] == 1)
321     {
322         _settextposition(Track_Channel + 7,2);
323
324         /* printf("\nChannel %i in track mode",I_channel);*/
325         track_threshold[I_channel] = Low_threshold;
326     }
327     Out_of_track_count[I_channel] = 0;
328     first_time_in_track[I_channel] = 0;
329
330 }
331 else
332 {
333     if (Out_of_track_count[I_channel] > 5)
334     {
335         Track_Flag[I_channel] = 0;
336         first_time_in_track[I_channel] = 1;
337         track_threshold[I_channel] = High_threshold;
338
339         /* Print out og track condition */
340         if (Out_of_track_count[I_channel] == 6)
341         {
342             _settextposition(Track_Channel + 7,2);
343             /* printf("\nChannel %i out of track",I_channel);*/
344         }
345     }
346
347     Out_of_track_count[I_channel] ++; /* Increment out of track counter */
348
349     V_signal = 0; /* Set U and V errors to zero so motors will */
350     U_signal = 0; /* not be command to move on bad data */
351
352 }
353
354 In_track = Track_Flag[I_channel];
355
356 Read_Actual_Pos(Channel_Address[I_channel],Act_pos);
357 I_stepB = Offset_Position[0][I_channel] - Act_pos[0];
358 I_stepA = Act_pos[1] - Offset_Position[1][I_channel];
359
360 /* Call Avtec control software for new position commands */
361 POSITION(&Track_Channel, &In_track, &U_signal,&V_signal,&I_stepA,&I_stepB);
362
363 /* Set up the commanded position for the DiskB Motor */
364 Motor_Command[0] = Offset_Position[0][I_channel] - I_stepB;
365
366 /* Set up the commanded position for the DiskA Motor */
367 Motor_Command[1] = Offset_Position[1][I_channel] + I_stepA;
368
369 Write_Final_pos(Channel_Address[I_channel],Motor_Command);
```



```
370     Write_to_flag_register (Channel_Address[I_channel],0x0808);
371
372     /* Output data to file if requested */
373     if (Record_Flag[I_channel] == 1)
374     {
375
376         fprintf(Data_Ptr," \n%6u, %8.3f, %8.3f, %8.3f, %9li, %9li, %9li, %9li, %9li, %9li",
377             Channel[I_channel - 1].Time_tag,
378             Channel[I_channel - 1].Signal_strength/204.7,
379             V_signal * 1000000,U_signal * 1000000,
380             Present_Position[0],Motor_Command[0],I_stepB,
381             Present_Position[1],Motor_Command[1],I_stepA);
382
383         Number_of_Samples ++; /* Record number of samples taken */
384
385         if (Number_of_Samples >= 1201)
386         {
387             Record_Flag[I_channel] = 0;
388             Number_of_Samples = 0;
389             printf("\nMax samples taken !!");
390         }
391
392     } /* End of if record flag = 1 */
393
394 } /* End of if Channel.Time_tag == Time_Tag */
395 else
396 {
397     Sync_Serial_Link();
398     break;
399 }
400
401     /* Send out command to toggle line that frame is done */
402     Port_Data = Port_Data & 0xFE;
403     outp(Port_C_Address,Port_Data); /* Send out command to initiate sync pulses */
404
405     Check_for_Key_Pressed();
406
407 } /* End of for I_channel = 1 to 3 */
408
409 } /* End of while = 1 */
410
411 } /* End of main function */
412
413
414 /* Function Init_Channel_Address of motor controller assignment*/
415 void Init_Channel_Address(void)
416 {
417     Channel_Address[1] = Motor_Pair1_Address;
418     Channel_Address[2] = Motor_Pair2_Address;
419     Channel_Address[3] = Motor_Pair3_Address;
420 }
421
422 /* Close_Down function will shut down the system in a orderly fashion */
423 void Close_down(void)
424 {
425     int Ch;
426
427     ComCloseAll(); /* Close all serial ports */
428
429     for (Ch = 1; Ch <= 3; Ch ++ )
430     {
431         /* Disable sync pulse to all channels */
432         Port_Data = Port_Data & Disable_Sync_Pulse[Ch];
433         outp(Port_C_Address,Port_Data);
434     }
435
436     /* Turn off 24 Volt power supply */
437     Port_Data = Port_Data & Pwr_24Volts_Off;
438     outp(Port_C_Address,Port_Data);
439
440     /* Set all position registers to actual position to stop motors */
441     for (Ch = 1; Ch <= 3; Ch++)
442         Set_Motor_Position(Channel_Address[Ch],Set_to_actual);
443
```

```
444     Write_system_data();
445
446     fclose (output_ptr);    /* Close file when done */
447     fclose (Data_Ptr);
448     fclose (System_Ptr);
449     fclose (AzEl_Ptr);
450     fclose (Align_Ptr);
451     fclose (Track_AzEl_Ptr);
452     exit (0);
453 }
454
455
456 /* Function To_Starting_Pos moves each channel to
457    the required position for tracking mode operation */
458
459 void Move_Chan_To_Starting_Pos(void)
460 {
461     long int Initial_Pos[3];
462     long int Target_Position[2][4];
463     long int Target[3];
464     int At_Target = 0;
465     int Delta_Limit = 25;    /* Delta between target and actual move to
466                               indicate the arm is at its designated
467                               position */
468
469     for (Track_Channel = 1; Track_Channel <= Number_of_Channels_To_Run; Track_Channel++)
470     {
471         I_channel = Channel_to_Run[Track_Channel];
472
473         V_signal = 0;
474         U_signal = 0;
475
476         /* Call control software to get initial position to were arm should go */
477         POSITION(&Track_Channel, &In_track, &U_signal,&V_signal,&I_stepA,&I_stepB);
478
479         /* Set up the commanded position for the Disk B Motor */
480         Motor_Command[0] = Offset_Position[0][I_channel] - I_stepB;
481         Target_Position[0][I_channel] = Motor_Command[0];
482
483         /* Set up the commanded position for the Disk A Motor */
484         Motor_Command[1] = Offset_Position[1][I_channel] + I_stepA;
485         Target_Position[1][I_channel] = Motor_Command[1];
486
487         /* Write out the new motor positions that we will need to move to */
488         Write_Final_pos(Channel_Address[I_channel],Motor_Command);
489
490         /* Start Trapizodial move */
491         Write_to_flag_registor (Channel_Address[I_channel],0x0808);
492
493     }
494     printf("\nChannels being sent to their starting positions");
495
496     /* Wait for channels to reach their destinations */
497     for (Track_Channel = 1; Track_Channel <= Number_of_Channels_To_Run; Track_Channel++)
498     {
499         I_channel = Channel_to_Run[Track_Channel];
500
501         Target[1] = Target_Position[1][I_channel];
502         Target[0] = Target_Position[0][I_channel];
503
504         At_Target = 0;
505
506         while (At_Target != 1)
507         {
508             Check_for_Key_Pressed();
509             At_Target = Reached_Commanded_Pos(Channel_Address[I_channel],Target,I_channel,Delta_Limit);
510         }
511     }
512
513     printf("\nChannels have reached their starting positions");
514
515 } /* End of function Move_Channels_To_Start */
516
517
```

```
518  /* Function Check_for_Key_Pressed (void) will be used check for an exit key */
519 void Check_for_Key_Pressed(void)
520 {
521     char Key_number;
522
523     if (kbhit())
524     {
525         Key_number = getch();
526
527         switch(Key_number)
528         {
529             case 'e': Close_Down();
530
531             default: break;
532
533         } /* End of switch statement */
534
535     } /* End of if kbhit() */
536
537 } /* End of function Check_for_Key_Pressed */
538
539 void Sync_Serial_Link(void)
540 {
541     /* Run through one cycle of sync pulses to synchronize communications */
542     for (Track_Channel = 1; Track_Channel <= Number_of_Channels_To_Run;
543         Track_Channel++)
544     {
545         I_channel = Channel_to_Run[Track_Channel];
546
547         while (I_channel != Check_for_Active_Sync_Pulse());
548
549         while (I_channel == Check_for_Active_Sync_Pulse());
550
551         Aquire_New_Time_Tag[I_channel] = 1;
552
553     }
554
555     ComFlushRx(); /* Flush out data before starting loop after synchronizing
556                  serial link */
557 }
558
559 void Compute_Offset(int I_chan)
560 {
561     double Max_Sig;
562     int Number_of_Max;
563     int loop;
564
565     Inter_Count[I_chan]++;
566
567     if ((Inter_Count[I_chan] >= 0) && (Inter_Count[I_chan] <= 4))
568     {
569         Uoffset[I_chan] = Uoffset[I_chan] + U_Delta[Inter_Count[I_chan]];
570         Voffset[I_chan] = Voffset[I_chan] + V_Delta[Inter_Count[I_chan]];
571     }
572     else
573     {
574         Inter_Count[I_chan] = 0;
575         Max_Sig = 0;
576
577         /* Determine max power signal for x samples */
578         for (loop = 0; loop <= 4; loop++)
579         {
580             if (Signal_Pwr[I_chan][loop] > Max_Sig)
581             {
582                 Max_Sig = Signal_Pwr[I_chan][loop];
583                 Number_of_Max = loop;
584             }
585         } /* End of for loop 0 to 4 */
586
587         if (I_chan == 2)
588             fprintf(Data_Ptr, "\n%6u, %8.3f, %8.3f, %8.3f, %8.3f, %8.3f, %8.3f, %8.3f, %8.3f, %8.3f",
589                 Channel[I_chan - 1].Time_tag, Signal_Pwr[I_chan][0],
```

```
592         Signal_Pwr[I_chan][1],Signal_Pwr[I_chan][2],
593         Signal_Pwr[I_chan][3],Signal_Pwr[I_chan][4],
594         Max_Sig, Number_of_Max);
595
596
597     Uoffset[I_chan] = Uoffset[I_chan] + MaxU_Delta[Number_of_Max];
598     Voffset[I_chan] = Voffset[I_chan] + MaxV_Delta[Number_of_Max];
599
600     /* Reinitialize signal strength array back to zero */
601     for (loop = 0; loop <= 5; loop++)
602         Signal_Pwr[I_chan][loop] = 0;
603
604     } /* End of Inter_Count[I_chan] == 5 */
605
606 } /* End of function Compute_offset */
607
608
609 void Init_Array(void)
610 {
611     int i;
612     for (i = 0; i<=5; i++)
613     {
614         U_Delta[i] = U_Delta[i] * Step_size;
615         V_Delta[i] = V_Delta[i] * Step_size;
616         MaxU_Delta[i] = MaxU_Delta[i] * Step_size;
617         MaxV_Delta[i] = MaxV_Delta[i] * Step_size;
618     }
619 }
620
621
622 void Wait_for_key_Press(void)
623 {
624     printf("\n Press any key to continue");
625     /* Display message until key is pressed. */
626
627     while( !kbhit()); /* Wait for key on keyboard to be pressed */
628
629     /* Use getch to throw key away. */
630     while (kbhit()) getch();
631
632 }
633
634
635
636
637
638
639
```

1	5.5	! Predicted azimuth of LEO 1 (deg)
2	3.0	! Predicted elevation of LEO 1 (deg)
3	-7.5	! Predicted azimuth of LEO 2 (deg)
4	4.0	! Predicted elevation of LEO 2 (deg)
5	8.0	! Predicted azimuth of LEO 3 (deg)
6	-3.5	! Predicted elevation of LEO 3 (deg)

```
1 .02          ! Duration of simulation in hours (REAL*8)
2 3            ! Number of disc channels (INTEGER)
3 40           ! Tracking rate in Hz (INTEGER)
4 0.5          ! Tracking gain (REAL*8)
5 .FALSE.      ! Write flag for detailed output (LOGICAL)
6 42164.       ! HEO orbital elements at time 0 (REAL*8)
7 0.
8 0.
9 0.
10 0.
11 0.
12 3            ! Number of LEO satellites (INTEGER)
13 6528.       ! LEO #1 orbital elements at time 0 (REAL*8)
14 0.
15 30.
16 25.
17 20.
18 0.
19 6878.       ! LEO #2 orbital elements at time 0 (REAL*8)
20 0.
21 28.
22 345.
23 310.
24 0.
25 6528.       ! LEO #3 orbital elements at time 0 (REAL*8)
26 0.
27 30.
28 330.
29 0.
30 0.
```

```

1  .module/ram/boot=0    adsys;
2
3  {#####}
4
5      {Module ad_conv will be used to for all A/D convertor related calls }
6
7      {Number of samples per frame}
8      .external          Points_per_frame;
9
10     {Counter used to track the number od data points taken}
11     .external          Data_Ready_to_be_Processed;
12
13     {Counts number of convert pulses to A/D}
14     .external          Convert_pulse_count;
15
16
17     {#####}
18     {Send_convert will be used to generate the convert pulse to the A/D}
19
20     .ENTRY send_convert;
21
22     send_convert:
23
24         ENA Sec_Reg;      {Select secondary registers}
25
26         ax1 = dm(Points_per_Frame);
27         ay1 = dm(Convert_pulse_count);
28
29         ar = ax1 - ay1;
30
31         if eq jump Stop_convert;      {Test to see if enough points}
32                                         {have been processes if so exit interrupt}
33
34
35         PM(14,M7) = ax0;              {Write out to the program memory to}
36                                         {initiate conversion of analog data}
37                                         {A low pulse of 80+ Nsec. will be }
38                                         {present at the four channel A/D}
39
40         ar = ay1 + 1;                  {Increment the number of pulse counts}
41                                         {in present frame}
42
43         dm(Convert_pulse_count) = ar;
44
45     Stop_convert:
46
47         DIS Sec_Reg;      {Switch registors back to primary mode}
48
49         rti;
50
51
52     {#####}
53
54     {Initialize_AD initializes A/D with doing one read from the A/D}
55
56     .ENTRY Initialize_AD;
57
58     Initialize_AD:
59
60         ay0 = PM(17,M7);
61
62         rts;
63
64     {Declare array that for storage of raw data from A/D All four channels}
65     {will be stored in array. Assignment is as follows:}
66     {Channel 1 will store in elements 1,5,9...}
67     {Channel 2 will store in elements 2,6,10...}
68     {Channel 3 will store in elements 3,7,11...}
69     {Channel 4 will store in elements 4,8,12...}
70     }
71
72     {#####}
73

```

```
74      {Read_data_int services interrupt from A/D
75      & reads data from all four A/D channels}
76 .ENTRY Read_data_int;
77
78 Read_data_int:
79
80     ENA Sec_Reg;      {Select secondary registers}
81
82     se = 4;           {Set up se register for multifunction shift inst}
83
84     ar = PM(17,M7);    {Read 1st channel of data from A/D}
85     sr = ASHIFT ar BY 4 (HI); {Shift data over to right}
86     sr = ASHIFT sr1 BY -4 (HI); {to extend sign bit and rescale back}
87                                {to proper scale and store data into array}
88
89     ar = PM(17,M7);    {Read 2nd channel of data from A/D}
90     DM(16,M5) = sr1, sr = ASHIFT ar (HI); {Shift data over to right to extend sign}
91     sr = ASHIFT sr1 BY -4 (HI); {bit and shift back left to rescale & store data
92                                for channel 1 in array}
93
94     ar = PM(17,M7);    {Read 3rd channel of data from A/D}
95     DM(16,M5) = sr1, sr = ASHIFT ar (HI); {Shift data over to right to extend sign}
96     sr = ASHIFT sr1 BY -4 (HI); {bit and shift back to left to rescale &
97                                store data for channel 2 in array}
98
99
100    ar = PM(17,M7);    {Read 4th channel of data from A/D}
101    DM(16,M5) = sr1, sr = ASHIFT ar (HI); {Shift data over to right to extend sign}
102    sr = ASHIFT sr1 BY -4 (HI); {bit and shiftback left to rescale & store data}
103                                {for channel 3 in array}
104
105    DM(16,M5) = sr1;    {Store data for channel 4 in array}
106
107    ay1 = dm(Data_Ready_to_be_Processed); {Increment number of points read in}
108    ar = ay1 + 1;
109    dm(Data_Ready_to_be_Processed) = ar;
110
111    DIS Sec_Reg;        {Switch registers back to primary mode}
112
113    rti;
114
115 .endmod;
```



```

1  .module/ram/boot=0 Algorithm;
2
3  {#####}
4
5  {Module algorithm will be used to compute the error signal for all four
6   quadrants on the detector}
7
8  {Total number of samples that will be stored at any one time}
9
10 {number_of_samples = 40 used for eprom version 40 samples
11  equals 10 samples of data per quadrant}
12 .const          number_of_filter_samples = 40;
13
14 {number_of_samples = 20000 used for emulator version 200 samples
15  equals 500 samples of data per quadrant}
16 (.const          number_of_filter_samples = 2000;)}
17
18 {Declare array that for storage of the outputs from the bandpass filter
19  will be stored in array. Assignment is as follows:
20  Channel 1 will store in elements 1,5,9...
21  Channel 2 will store in elements 2,6,10...
22  Channel 3 will store in elements 3,7,11...
23  Channel 4 will store in elements 4,8,12...
24  }
25
26 {Array to store the output of the bandpass filter}
27 .var/dm/ram/circ      Filtered_data[number_of_filter_samples];
28 .global Filtered_data;
29
30 {Array to store the output of the bandpass filter}
31 .var/dm/ram/circ      Lowpass_Filt_data[number_of_filter_samples];
32 .global Lowpass_Filt_data;
33
34 {Pointer,length, and count increment of array for bandpass filter}
35 .var/dm/ram          Filtered_data_add_ptr;
36 .var/dm/ram          Filtered_data_len_ptr;
37 .var/dm/ram          Filtered_data_count_ptr;
38
39 .global              Filtered_data_add_ptr;
40 .global              Filtered_data_len_ptr;
41
42 {Pointer,length, and count increment of array for Lowpass filter}
43 .var/dm/ram          Low_Filt_data_add_ptr;
44 .var/dm/ram          Low_Filt_data_len_ptr;
45 .var/dm/ram          Low_Filt_data_count_ptr;
46
47 .global              Low_Filt_data_add_ptr;
48 .global              Low_Filt_data_len_ptr;
49
50 .var/dm/ram          Number_of_Data_Points_Processed; {Counter for tracking
51                                                         number of points processed}
52
53 .global              Number_of_Data_Points_Processed; {Counter for tracking
54                                                         number of points processed}
55
56 {The following variables will be the final results from a frames worth of
57  data }
58
59 .var/dm/ram          Total_signal_strength;
60 .var/dm/ram          Elevation_error;
61 .var/dm/ram          Azimuth_error;
62
63 .global              Total_signal_strength;
64 .global              Elevation_error;
65 .global              Azimuth_error;
66
67 .external            Process_data_add_ptr;
68 .external            Process_data_len_ptr;
69 .external            Bandpass_Filt;
70 .external            Lowpass_Filt;
71
72 .external            Delay_1;      {Bandpass filter delays for channel 1}
73 .external            Delay_2;      {Bandpass filter delays for channel 2}

```

```

74 .external      Delay_3;      {Bandpass filter delays for channel 3}
75 .external      Delay_4;      {Bandpass filter delays for channel 4}
76
77 .external      Pointer_1;     {Bandpass filter delays for channel 1}
78 .external      Pointer_2;     {Bandpass filter delays for channel 2}
79 .external      Pointer_3;     {Bandpass filter delays for channel 3}
80 .external      Pointer_4;     {Bandpass filter delays for channel 4}
81
82 .external      Delay_1L;      {Lowpass filter delays for channel 1}
83 .external      Delay_2L;      {Lowpass filter delays for channel 2}
84 .external      Delay_3L;      {Lowpass filter delays for channel 3}
85 .external      Delay_4L;      {Lowpass filter delays for channel 4}
86
87 .external      Pointer_1L;     {Lowpass filter delays for channel 1}
88 .external      Pointer_2L;     {Lowpass filter delays for channel 2}
89 .external      Pointer_3L;     {Lowpass filter delays for channel 3}
90 .external      Pointer_4L;     {Lowpass filter delays for channel 4}
91
92
93 .ENTRY Filter_error_signal;
94
95 Filter_error_signal:
96
97     {Filter channel 1 thru bandpass}
98
99         L5 = %Delay_1;          {Set up the length of the filter
100                                delay array for the FIR bandpass filter}
101
102         I5 = DM(Pointer_1);     {15 points to delay elements }
103
104         MY1 = DM(I0,M0);        {Send channel 1 data to input of
105                                Bandpass filter}
106
107         Call Bandpass_filt;     {Run data point from channel 1
108                                thru bandpass filter}
109
110         DM(Pointer_1) = I5;     {Store pointer of last sample to be
111                                saved}
112
113         DM(I2,M0) = MR1;        {Store output from filter}
114
115     {Filter channel 2 thru bandpass}
116
117         I5 = DM(Pointer_2);     {15 points to delay elements }
118
119         MY1 = DM(I0,M0);        {Send channel 2 data to input of
120                                Bandpass filter}
121
122         Call Bandpass_filt;     {Run data point from channel 2
123                                thru bandpass filter}
124
125         DM(Pointer_2) = I5;     {Store pointer of last sample to be
126                                saved}
127
128         DM(I2,M0) = MR1;        {Store output from filter}
129
130     {Filter channel 3 thru bandpass}
131
132         I5 = DM(Pointer_3);     {15 points to delay elements }
133
134         MY1 = DM(I0,M0);        {Send channel 3 data to input of
135                                Bandpass filter}
136
137         Call Bandpass_filt;     {Run data point from channel 3
138                                thru bandpass filter}
139
140         DM(Pointer_3) = I5;     {Store pointer of last sample to be
141                                saved}
142
143         DM(I2,M0) = MR1;        {Store output from filter}
144
145     {Filter channel 4 thru bandpass}
146
147         I5 = DM(Pointer_4);     {15 points to delay elements }

```

```
148
149
150      MY1 = DM(I0,M0);           {Send channel 4 data to input of
151                                {Bandpass filter}
152
153      Call Bandpass_filt;        {Run data point from channel 4
154                                {thru bandpass filter}
155
156      DM(Pointer_4) = I5;        {Store pointer of last sample to be
157                                {saved}
158
159      DM(I2,M0) = MR1;           {Store output from filter}
160
161      {Change I2 and L5 for lowpass filter computation}
162
163      Modify(I2,M3);             {Decrement the address pointer I2
164                                {by the value of M3}
165
166      L5 = %delay_1L;            {Set up length of FIR lowpass filter
167                                {delay line}
168
169      {Filter channel 1 thru lowpass filter}
170
171      I5 = DM(Pointer_1L);        {15 points to delay elements }
172
173      ax0 = DM(I2,M0);           {Send channel 1 data to input of}
174                                {Bandpass filter}
175      ar = abs ax0;              {Rectify signal by taking the
176                                {absolute value of ax0}
177      my1 = ar;                  {Input parameter to lowpass filter}
178
179      Call Lowpass_filt;         {Run data point from channel 1
180                                {thru bandpass filter}
181
182      DM(Pointer_1L) = I5;        {Store pointer of last sample to be
183                                {saved}
184
185      DM(I3,M0) = MR1;           {Store output from filter}
186
187      {Filter channel 2 thru lowpass}
188
189      I5 = DM(Pointer_2L);        {15 points to delay elements }
190
191      ax0 = DM(I2,M0);           {Send channel 2 data to input of}
192                                {Bandpass filter}
193      ar = abs ax0;              {Rectify signal by taking the
194                                {absolute value of ax0}
195      my1 = ar;                  {Input parameter to lowpass filter}
196
197      Call Lowpass_filt;         {Run data point from channel 2
198                                {thru bandpass filter}
199
200      DM(Pointer_2L) = I5;        {Store pointer of last sample to be
201                                {saved}
202
203      DM(I3,M0) = MR1;           {Store output from filter}
204
205      {Filter channel 3 thru lowpass}
206
207      I5 = DM(Pointer_3L);        {15 points to delay elements }
208
209      ax0 = DM(I2,M0);           {Send channel 3 data to input of}
210                                {Bandpass filter}
211      ar = abs ax0;              {Rectify signal by taking the
212                                {absolute value of ax0}
213      my1 = ar;                  {Input parameter to lowpass filter}
214
215      Call Lowpass_filt;         {Run data point from channel 3
216                                {thru bandpass filter}
217
218      DM(Pointer_3L) = I5;        {Store pointer of last sample to be
219                                {saved}
220
221      DM(I3,M0) = MR1;           {Store output from filter}
```

```

222 {Filter channel 4 thru lowpass}
223
224     15 = DM(Pointer_4L);           {15 points to delay elements }
225
226     ax0 = DM(12,M0);               {Send channel 4 data to input of
227                                     Bandpass filter}
228     ar = abs ax0;                   {Rectify signal by taking the
229                                     absolute value of ax0}
230     my1 = ar;                       {Input parameter to lowpass filter}
231
232     Call Lowpass_filt;              {Run data point from channel 4
233                                     thru bandpass filter}
234
235     DM(Pointer_4L) = 15;           {Store pointer of last sample to be
236                                     saved}
237
238     DM(13,M0) = MR1;               {Store output from filter}
239
240     {Increment the number of data points processed through algorithm}
241
242     ay1 = dm(Number_of_Data_Points_Processed);
243     ar = ay1 + 1;
244     dm(Number_of_Data_Points_Processed) = ar;
245
246     rts;
247
248 .ENTRY Compute_Tracking_Error;
249
250 Compute_Tracking_Error:
251
252     Modify (i3,m3);                {Move address pointer back four places to pull
253                                     out last computed values from frame}
254
255     ax0 = Dm(i3,m0);               {Store quadrant A data (channel 1) in ax0}
256     ay0 = Dm(i3,m0);               {Store quadrant B data (channel 2) in ay0}
257     ax1 = Dm(i3,m0);               {Store quadrant C data (channel 3) in ax1}
258     ay1 = Dm(i3,m0);               {Store quadrant D data (channel 4) in ay1}
259
260
261     {Calculate total signal strength}
262     ar = ax0 + ay0;                 {Compute Quadrant A + B magnitude}
263     af = ax1 + ay1;                 {Compute Quadrant C + D magnitude}
264
265     ar = ar + af;                   {Sum all Quadrants A + B + C + D }
266
267     DM(Total_signal_strength) = ar;
268
269
270     {Compute Azimuth error}
271
272     ar = ax0 + ay0;                 {Compute Quadrant A + B magnitude}
273
274     ar = ar - af;                   {Subtract Quadrants A + B - C - D where
275                                     af = C + D}
276
277     DM(Azimuth_error) = ar;
278
279
280     {Compute elevation}
281
282     ar = ax0 + ay1;                 {Add Quadrant A + D}
283
284     af = ax1 + ay0;                 {Add Quadrant C + B}
285
286     ar = ar - af;                   {Subtract Quadrant (A + D) - (C + B) }
287
288     DM(Elevation_error) = ar;       {Store Azimuth error}
289
290     RTS;
291
292 .ENDMOD;

```

```

1 {
2 *****
3 * Analog Dev. ADSP-210x Filter Realization Copyright (C) by HYPERCEPTION, INC.
4 *****
5 *
6 *     Filter Generated from File: kwband.FIR
7 *
8 *     Filter Order=42
9 *
10 *     Direct Form Realization of the following convolution sum:
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 *
25 *
26 *
27 *
28 *
29 *
30 *
31 *****
32 *
33 *
34 {*
35 *
36 *     The following main module example may be copied to a separate
37 *     filename (e.g. Main.dsp) and uncommented to provide a test of
38 *     the filter. A typical example of assembly and link commands
39 *     needed to produce executable filter test code would then be:
40 *
41 *     dsppa Main <cr>
42 *     dsppa kwband <cr>
43 *     dsppl Main kwband -a archname.ACH -g <CR>
44 *
45 *     NOTE: The Linker expects an Architecture Description File (.ACH)
46 *     which is created by the System Builder.
47 *}
48 .MODULE /RAM/boot=0 FIR_Filt;      { Q15-Notation (scaling) is in effect }
49 .CONST Length=42;                { Length of impulse response }
50
51 .VAR/PM/CIRC Delay_1[Length];
52 .VAR/PM/CIRC Delay_2[Length];
53 .VAR/PM/CIRC Delay_3[Length];
54 .VAR/PM/CIRC Delay_4[Length];
55
56 .Global Delay_1;
57 .Global Delay_2;
58 .Global Delay_3;
59 .Global Delay_4;
60
61 .VAR/DM Pointer_1[1];
62 .VAR/DM Pointer_2[1];
63 .VAR/DM Pointer_3[1];
64 .VAR/DM Pointer_4[1];
65
66 .global Pointer_1;
67 .global Pointer_2;
68 .global Pointer_3;
69 .global Pointer_4;
70
71 .VAR/DM Coefficients[Length];
72 .global Coefficients;
73

```

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

[illegible]

```
148          DO init_band_coef UNTIL CE;
149
150          sr1 = PM(I5,M5);
151          sr0 = px;      {Store 8 data bits from px into sr0}
152          sr = sr or lshift sr1 by -8 (HI);
153
154  init_band_coef: DM(I1,M1) = sr0;
155
156          RTS;
157
158  .ENDMOD;
159
```

```

1  {
2  *****
3  * Analog Dev. ADSP-210x Filter Realization Copyright (C) by HYPERCEPTION, INC.
4  *
5  *
6  *     Filter Generated from File: kwlow.FIR
7  *
8  *     Filter Order=45
9  *
10 *     Direct Form Realization of the following convolution sum:
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 *
25 *
26 *
27 *
28 *
29 *
30 *
31 *****
32 *
33 }
34
35 .MODULE /ram/boot = 0 LOW_FIR_Filt;    ( Q15-Notation (scaling) is in effect )
36 .CONST Length_low=45;                ( Length of impulse response )
37
38 {Declare delay line arrays for lowpass filter}
39
40 .VAR/PM/CIRC Delay_1L[Length_low];
41 .VAR/PM/CIRC Delay_2L[Length_low];
42 .VAR/PM/CIRC Delay_3L[Length_low];
43 .VAR/PM/CIRC Delay_4L[Length_low];
44
45 .Global Delay_1L;
46 .Global Delay_2L;
47 .Global Delay_3L;
48 .Global Delay_4L;
49
50 {Declare pointers that will point to the delay line array elements}
51
52 .VAR/DM Pointer_1L[1];
53 .VAR/DM Pointer_2L[1];
54 .VAR/DM Pointer_3L[1];
55 .VAR/DM Pointer_4L[1];
56
57 .global Pointer_1L;
58 .global Pointer_2L;
59 .global Pointer_3L;
60 .global Pointer_4L;
61
62 .VAR/DM Coefficients_low[Length_low];
63 .Global Coefficients_low;
64
65 .VAR/PM Low_Coeff[Length_low];
66 .Global Low_Coeff;
67
68 .INIT Low_Coeff: 26,35,58,88,127,176,236,
69                  306,387,477,577,683,795,909,
70                  1023,1134,1239,1333,1416,1483,1532,
71                  1563,1573,1563,1532,1483,1416,1333,
72                  1239,1134,1023,909,795,683,577,
73                  477,387,306,236,176,127,88,

```


[illegible]

```

1 .module/ram/boot=0/abs=0 MAIN;
2
3 {#####}
4
5 {Module main will be used to initialize all system parameters }
6 {and service all interrupts}
7
8
9 .include <c:\nasa\dspcode\ezlab\def2101.h>;          ( file has sys. cntl. registers )
10
11 {Total number of samples that will be stored at any one time}
12
13 { Used for emulator version }
14 (.const                number_of_samples = 2000;)
15
16 { Used for Eprom version }
17 .const                number_of_samples = 400;
18
19 {Declare array that for storage of raw data from A/D All four channels
20 will be stored in array. Assignment is as follows:
21 Channel 1 will store in elements 1,5,9...
22 Channel 2 will store in elements 2,6,10...
23 Channel 3 will store in elements 3,7,11...
24 Channel 4 will store in elements 4,8,12...
25 }
26 .var/dm/ram/circ      AD_Raw_Data[number_of_samples];
27
28 {Pointer,length, and count increment of array that receives data from A/D)
29 .var/dm/ram           AD_rawdata_add_ptr;
30 .var/dm/ram           AD_rawdata_len_ptr;
31 .var/dm/ram           AD_rawdata_count_ptr;
32
33 {Points to the element in the array of raw data that is being processed}
34 .var/dm/ram           Process_data_add_ptr;
35 .global               Process_data_add_ptr;
36 .var/dm/ram           Process_data_len_ptr; {Length of array}
37 .global               Process_data_len_ptr;
38
39 {Counter used to track the number od data points taken}
40 .var/dm/ram           Data_Ready_to_be_Processed;
41 .global               Data_Ready_to_be_Processed;
42
43 {Maximum number of data points to be taken per frame}
44 .var/dm/ram           Points_per_Frame;
45 .global               Points_per_Frame;
46 {.init Points_per_Frame: 100;} {Number of samples of data per frame}
47
48 {Counts number of convert pulses to A/D}
49 .var/dm/ram           Convert_pulse_count;
50 .global               Convert_pulse_count;
51
52 .var/dm/ram           count_irq0;
53 .var/dm/ram           Read_adc_add_ptr;
54 .var/dm/ram           Convert_pulse_add_ptr;
55 .var/dm/ram           Sync_flag_enable;
56
57 {Time tag for data to be sent over to host computer}
58 .var/dm/ram           Frame_tag;
59 .global               Frame_tag;
60 {.init Frame_tag: 000; }
61
62 {Memory mapped I/O declarations}
63 .port                 read_adc;    {Mapped I/O address that A/D is read}
64 .global               read_adc;
65
66 .port                 convert_pulse; {Mapped I/O address that initiates}
67 .global               convert_pulse; {conversion on A/D}
68
69 .port                 watch_dog;   {Mapped I/O address to reset watchdog timer}
70 .global               watch_dog;
71
72 .external              Number_of_Data_Points_Processed; {Counter for tracking
73                  number of points processed}

```

```

74
75 .external      Delay_1;      {Bandpass filter delays for channel 1}
76 .external      Delay_2;      {Bandpass filter delays for channel 2}
77 .external      Delay_3;      {Bandpass filter delays for channel 3}
78 .external      Delay_4;      {Bandpass filter delays for channel 4}
79
80 .external      Pointer_1;     {Bandpass filter delays for channel 1}
81 .external      Pointer_2;     {Bandpass filter delays for channel 2}
82 .external      Pointer_3;     {Bandpass filter delays for channel 3}
83 .external      Pointer_4;     {Bandpass filter delays for channel 4}
84
85 .external      Delay_1L;      {Lowpass filter delays for channel 1}
86 .external      Delay_2L;      {Lowpass filter delays for channel 2}
87 .external      Delay_3L;      {Lowpass filter delays for channel 3}
88 .external      Delay_4L;      {Lowpass filter delays for channel 4}
89
90 .external      Pointer_1L;     {Lowpass filter delays for channel 1}
91 .external      Pointer_2L;     {Lowpass filter delays for channel 2}
92 .external      Pointer_3L;     {Lowpass filter delays for channel 3}
93 .external      Pointer_4L;     {Lowpass filter delays for channel 4}
94
95 .external      Filtered_data_add_ptr;
96 .external      Filtered_data_len_ptr;
97 .external      Filtered_data;
98
99 .external      Coefficients;
100
101 .external      Low_Filt_data_add_ptr;
102 .external      Low_Filt_data_len_ptr;
103 .external      Lowpass_Filt_data;
104
105 .external      Coefficients_low;
106
107 {Declarations of external subroutines in other modules}
108 .external      Filter_error_signal; {Subroutine for filtering}
109                                     {error signal}
110 .external      Initialize_AD;      {Subroutine to initialize A/D}
111
112 .external      Send_convert;        {Subroutine used to generate
113                                     convert pulse to A/D convertor}
114
115 .external      Read_Data_Int;       {Subroutine reads in all 4 channels
116                                     of data from the A/D and sign extends
117                                     the data}
118
119 .external      Compute_Tracking_Error; {Subroutine to compute error signals}
120
121 .external      Transmit_data;       {Subroutine used to transmitt data to
122                                     host computer}
123
124 .external      Send_data;           {Subroutine used to service
125                                     sport0 transmit interrupt}
126
127 .external      Init_Lowpass_Coeff;  {Subroutine used to
128                                     Initialize Lowpass coefficients}
129
130 .external      Init_Bandpass_Coeff; {Subroutine used to
131                                     Initialize Bandpass coefficients}
132
133 .external      Set_up_uart_parameters; {Subroutine used to
134                                     Initialize uarts parameters}
135
136 {Listed below is the interrupt jump table for the 2101}
137 {Resetting flag_out disables the rs232 driver chip }
138
139      jump start; nop; nop; nop;      { reset vector}
140      jump read_data_int; nop; nop; nop; { irq2 interrupt vector}
141      jump Send_data; nop; nop; nop;    { sport0 transmit vector}
142      rti; nop; nop; nop;               { sport0 receive vector}
143      jump send_convert; nop; nop; nop; { irq1 interrupt vector}
144
145      set flag_out;                     { irq0 interrupt vector}
146      call Reset_Control_Flags;
147      call Sync_pulse_disable;

```

```

148         rti;
149
150         DM(Watch_dog) = ax0;                ( timer interrupt vector)
151         { reset flag_out;}                 ( used for sync mode)
152         rti;
153         nop;
154         nop;
155
156         {call Reset_Control_Flags; rti; nop; nop;} (timer interrupt
157                                                     used for non sync mode)
158         {End of the interrupt jump table}
159
160     start:
161
162         call Init_sys_par;                  (Initialize system parameters)
163
164         call Init_Lowpass_Coeff;            (Initialize Lowpass coefficients)
165
166         call Init_Bandpass_Coeff;           (Initialize Bandpass coefficients)
167
168         call Set_Primary_Data_Reg;          (Initialize all data registers)
169
170         call Set_up_uart_parameters;        (Initialize uarts parameters)
171
172         call Initialize_AD;                 (Initialize A/D for operation)
173
174         call Reset_Control_Flags;           (Reset counters for number of samples
175                                             taken and processed)
176
177         reset flag_out;                     (Disable serial port driver)
178
179         ax1 = dm(Points_Per_Frame);         (Set points_per_frame and)
180         dm(Convert_Pulse_Count)= ax1;       (Convert_pulse_count equal)
181
182         ax0 = 1;
183         dm(Sync_flag_enable) = ax0;
184
185         ifc = 0x3f;                         (Clear all interrupts before enabling )
186         imask = 0x37;                       (Enable interrupts Irq2,
187                                             Sport0 Transmit, Irq1, and Irq0 for
188                                             sync mode,timer)
189
190         {imask = 0x35;}                     (Enable interrupts Irq2,
191                                             Sport0 Transmit Irq1, and Timer for
192                                             non sync mode)
193
194         {Enable timer used for sync mode of operation it will be used to
195          reset the watchdog timer}
196
197         ENA Timer;                          (Start timer for watchdog timer updates)
198
199
200
201     main_loop:
202
203         { Wait for sync pulse to occur before continuing }
204         ax0 = dm(Sync_flag_enable);
205         ar = Pass ax0;
206         if ne jump main_loop;
207         imask = 0x35;                        (Disable interrupt IRQ0)
208
209     Inner_loop:
210         ax1 = dm(Data_Ready_to_be_Processed);
211         ay1 = dm(Number_of_Data_Points_Processed);
212
213         ar = ax1 - ay1;
214
215         if eq jump Inner_loop;               (Test and see if data needs to be processed
216                                             if not wait for next data to be input read from
217                                             A/D)
218
219         call Filter_Error_Signal;           (Call algorithm to filter error signal)
220
221         ax1 = dm(Points_per_Frame);

```

```

222     ay1 = dm(Number_of_Data_Points_Processed);
223
224     ar = ax1 - ay1;
225
226     if ne jump Inner_loop;           {Test to see if enough points have been processes}
227
228     call Compute_Tracking_Error;     {Compute azimuth & elevation error and total
229                                     signal strength}
230
231     call Transmit_data;              {Transmit data to Host computer}
232
233
234
235     DM(Watch_dog) = ax0;             {Write out to the data mememory to}
236                                     {indicate that the processor is}
237                                     {still running. A low pulse of 240 Nsec.}
238                                     {will be present at the watchdog timer}
239
240     DIS AR_SAT;
241
242     ay1 = dm(Frame_tag);             {Increment counter to indicate what frame
243                                     the software is on}
244
245     ar = ay1 + 1;
246     dm(Frame_tag) = ar;
247
248     Wait_for_enable:
249     ax0 = dm(Sync_flag_enable);
250     ar = Pass ax0;
251     if ne jump Wait_for_enable;
252
253     ifc = 0x02;
254     imask = 0x37;                   {Clear irq0 before enabling interrupt}
255                                     {Enable IRQ0}
256
257     ENA AR_SAT;
258
259     jump main_loop;
260
261     {#####}
262     {Disable_irq1 will disable irq1 for conversion}
263     {#####}
264     disable_irq1:
265     ay0 = 0x3b;
266     ax0 = imask;                   {Set ay0 for bit to mask of irq1}
267     ar = ax0 AND ay0;
268     imask = ar;
269     rts;
270
271
272
273     {#####}
274     {Disable_irq2 will disable irq1 for conversion}
275     {#####}
276     disable_irq2:
277     ay0 = 0x1f;
278     ax0 = imask;                   {Set ay0 for bit to mask of irq2}
279     ar = ax0 AND ay0;
280     imask = ar;
281     rts;
282
283
284
285     {#####}
286     {Disable_timer will disable timer for conversion}
287     {#####}
288     disable_timer:
289     dis timer;
290     rts;
291
292
293
294     {#####}
295     {Reset control flags used to reset counters}

```

```

296
297 .entry Reset_Control_Flags;
298
299 Reset_Control_Flags:
300
301     ar = 0;
302     dm(Data_Ready_to_be_Processed) = ar;
303     dm(Number_of_Data_Points_Processed) = ar;
304     dm(Convert_pulse_count) = ar;
305
306     rts;
307
308     {#####
309     {Resets timer function at beginning of 40 Hz. sample frame}
310
311     Reset_Timer:
312         ax0 = 7499;
313         dm(Tcount_Reg) = ax0;    {Set up timer to timeout after 6 millisec.}
314         nop;
315         Set_Flag_out;            {Enable RS232 driver for data transmission}
316         rts;
317
318     {#####
319     {Initializing subroutine for DSP specific registers}
320
321     Init_sys_par:
322
323         ax0 = 0x1819;
324         dm(Sys_Ctrl_Reg) = ax0;    {Sport 0 enabled, Sport 1 enabled &}
325                                     {config as FI,F0,Irq0,1,2, Boot page 0,}
326                                     {One wait states for boot memory}
327                                     {One wait states for program memory}
328
329
330         ax0 = 0x3200;
331         dm(Dm_Wait_Reg) = ax0;    {Set Data memory wait states to }
332                                     {the following settings DWAIT0,3,4=0}
333                                     {DWAIT2 = 1 wait state for reading}
334                                     {data from A/D DWAIT1 = 3 for }
335                                     {generating output pulse to watchdog}
336                                     { timer }
337
338         { Set up timer parameters}
339         ax0 = 9;
340         {ax0 = 39;}
341
342         dm(Tscale_Reg) = ax0;
343
344         ax0 = 30719;
345         dm(Tperiod_Reg) = ax0;    {Tscale_reg will be used for 40 hz. update}
346                                     {set to 39 to allow for PC to keep up}
347                                     {used for debug only sets update to .1 sec}
348                                     { decrement tcount every 10 th instruction cycle}
349
350         ax0 = 10000;
351                                     { This value in timer counter corresponds }
352                                     { to interrupts generated every 25 milliseconds}
353                                     { for the frame timer}
354
355         dm(Tcount_Reg) = ax0;
356                                     {Delay the first interrupt by approx.}
357                                     { 8.3 milli seconds}
358
359         {Initialize Sport1 control register}
360
361         ax0 = 0x4000;
362         dm(Sport1_Ctrl_Reg) = ax0;    {Internal clock of Sport1 enabled}
363
364
365         ax0 = 225;
366         dm(Sport1_Sclkdiv) = ax0;    {Set up timing for Sport1 serial clock}
367
368         {Initialize Sport0 control register}
369
370         ax0 = 0;
371
372         dm(SPORT0_RX_WORDS1) = ax0; {Disable Sport 0 Multichannel cabibility}
373         dm(SPORT0_RX_WORDS0) = ax0;
374         dm(SPORT0_TX_WORDS1) = ax0;

```

[illegible]

```

444 DM(Filtered_data_add_ptr) = I2;           {Sore address pointer}
445
446 L2 = %Filtered_data;                       {Set length of buffer for a }
447                                           {circular array}
448
449 DM(Filtered_data_len_ptr) = L2;           {Store length of output filter}
450                                           {buffer for bandpass filter}
451
452
453
454
455
456 I3 = `Lowpass_Filt_data;                   {Set up address pointer for array
457                                           of outputs from bandpass filter}
458
459 DM(Low_Filt_data_add_ptr) = I3;           {Store address pointer for pointing
460                                           to data in the array that will be processed}
461
462
463 L3 = %Lowpass_Filt_data;                   {Set length of buffer for lowpass
464                                           filter array}
465
466 DM(Low_Filt_data_len_ptr) = L3;           {Store length of output filter}
467                                           {buffer for lowpass filter}
468
469
470
471 {Set up address pointer for generating convert pulse}
472 I4 = `Convert_Pulse;
473 DM(Convert_Pulse_Add_ptr) = I4;
474
475
476
477 {Set up address of delay line of bandpass filter and store address
478   there is 1 delay line for each quadrant}
479
480 I5 = `Delay_1;
481 DM(Pointer_1) = I5;
482
483 I5 = `Delay_2;
484 DM(Pointer_2) = I5;
485
486 I5 = `Delay_3;
487 DM(Pointer_3) = I5;
488
489 I5 = `Delay_4;
490 DM(Pointer_4) = I5;
491
492 {Set up address of delay line of lowpass filter and store address
493   there is 1 delay line for each quadrant}
494
495 I5 = `Delay_1L;
496 DM(Pointer_1L) = I5;
497
498 I5 = `Delay_2L;
499 DM(Pointer_2L) = I5;
500
501 I5 = `Delay_3L;
502 DM(Pointer_3L) = I5;
503
504 I5 = `Delay_4L;
505 DM(Pointer_4L) = I5;
506
507 I5 = `Delay_1;   {Reset address of I5 to initial delay element that
508                  processed}
509
510 L5 = %Delay_1;   {L5 will be used as the delay element length for both
511                  bandpass and lowpass filter}
512
513
514
515 I6 = `AD_Raw_Data;           {Set up address pointer for array
516                               AD_Raw_data}
517

```



```
518     DM(AD_rawdata_add_ptr) = 16;           (Store address pointer for input array)
519
520
521     L6 = %AD_Raw_Data;                     (Set length of buffer for a
522                                           circular array)
523
524     DM(AD_rawdata_len_ptr) = L6;           (Store length on input array)
525
526
527
528     I7 = 'Read_Adc;                        (Set up address pointer for reading A/D)
529     DM(Read_Adc_Add_Ptr) = I7;           ( Store address )
530
531
532     M0 = 1;                               (Set up address counter for incrementing through the
533                                           Raw_data_array, and intermediate arrays after the
534                                           bandpass and lowpass filter)
535
536     M1 = 1;                               (Set address counter to be used for bandpass & lowpass
537                                           filters)
538
539     M3 = -4;                              (Set up address counter to -4 to decrement address
540                                           between bandpass and lowpass filtering)
541
542     M5 = 1;                               (Set address counter to be used for bandpass & lowpass
543                                           filters)
544
545     M6 = 2;                               (Set up address counter to increment delay storage
546                                           in bandpass and lowpass filter this provides the required
547                                           offset in storing the data)
548
549     M7 = 0;                               (Set up address counter to zero such that when
550                                           reading the A/D and Sending the convert pulse that we
551                                           don't change the address)
552
553     (The following counters and array length indicators are spare and will
554     be sent to zero)
555
556
557     M2 = 0;
558     M4 = 0;
559
560     L4 = 0;
561     L7 = 0;
562
563     rts;
564
565     .Entry Sync_pulse_enable;
566
567     Sync_pulse_enable:
568         ax0 = 1;
569         dm(Sync_flag_enable) = ax0; (Set up flag for enabling IRQ0)
570     rts;
571
572     Sync_pulse_disable:
573         ax0 = 0;
574         dm(Sync_flag_enable) = ax0; (Set up flag for disabling IRQ0)
575     rts;
576
577
578     Check_sync_flag:
579
580         ax0 = dm(Sync_flag_enable);
581         ar = Pass ax0;
582         if eq jump Disable_Irq0;
583         ifc = 0x20;                       (Clear irq0 before enabling interrupt)
584         imask = 0x37;                     (Enable IRQ0)
585     rts;
586
587     Disable_Irq0:
588         imask = 0x35;                     (Disable IRQ0)
589     rts;
590
591     .endmod;
```

```

1  .module/ram/boot=0    uart;
2
3  {#####}
4  .const buffer_length = 8;
5
6  {Output_Buffer will be used to store the formatted Data words}
7  .var/dm/ram/circ      Output_Buffer[buffer_length];
8
9  .var/dm/ram            Number_of_words_sent;
10
11 .var/dm/ram            Words_to_be_transmitted;
12
13 {Set up temporary buffers for storing IO,M0,L0}
14 .var/dm/ram            Temp_IO_buffer;
15 .var/dm/ram            Temp_L0_buffer;
16 .var/dm/ram            Temp_M0_buffer;
17
18 .external              Total_signal_strength;
19 .external              Elevation_error;
20 .external              Azimuth_error;
21 .external              Frame_tag;
22
23
24 {Declare external subroutines used}
25 .external              Reset_Control_Flags;    {Subroutine used to reset
26                                                  control flags}
27 .external              Sync_Pulse_Enable;      {Sets flag for enabling
28                                                  IRQ0 interrupt }
29
30 {Marco delay loop used to generate a delay .8 uSec per count}
31
32 .MACRO DELAY(%0);
33 .LOCAL Delay_loop;
34     cntr = %0;
35     DO delay_loop until ce;
36     nop; nop; nop; nop; nop;
37     nop; nop; nop; nop;
38     delay_loop: nop;
39 .ENDMACRO;
40
41 {#####}
42 {Transmit_data will be used to transmit data to the host computer}
43
44 .ENTRY Transmit_data;
45
46 Transmit_data:
47
48     ar = 0;                      {Reset counter to zero}
49     dm(Number_of_words_sent) = ar;
50
51     Call Set_address_pointers;    {Set up address pointers}
52
53     {Call Set_up_test_data;}      {Subroutine used for testing purposes
54                                   sets up predefined data for transmission}
55
56     Call Format_data;              {Format data into 8 bit words}
57
58     IO = ^Output_buffer;
59
60     ar = DM(IO,M0);               {Transmit first word to host PC}
61     TX0 = ar;
62
63     ay1 = dm(Number_of_words_sent);
64     ar = ay1 + 1;
65     dm(Number_of_words_sent) = ar;
66
67     RTS;
68
69 {#####}
70 {Set address pointer store primary array address and length and
71  temporary assigns new values to IO,L0,M0}
72
73 Set_address_pointers:

```

```

74
75     dm(Temp_IO_buffer) = IO;      (Store IO,L0, and M0)
76     dm(Temp_L0_buffer) = L0;
77     dm(Temp_M0_buffer) = M0;
78
79     IO = %Output_buffer;          (Set IO to address of output buffer)
80     L0 = %Output_buffer;          (Set L0 to length of output buffer)
81     M0 = 1;
82
83     RTS;
84
85
86     (#####)
87     (Reset address pointer store primary array address and length and
88     temporary assigns new values to IO,L0,M0 This subroutine is called
89     when Sport0 has transmitted all 8 data words)
90
91     Reset_address_pointer:
92
93         IO = dm(Temp_IO_buffer);    (Primary data back to IO,L0, and M0)
94         L0 = dm(Temp_L0_buffer);
95         M0 = dm(Temp_M0_buffer);
96
97         Delay(1000);                (Delay disabling driver by .8 mSec.)
98
99         reset_flag_out;             (Disable serial driver till next sync pulse)
100
101     RTS;
102
103
104     (#####)
105     (Format_data is used to format the data to 8 bit words for data transmission)
106
107     Format_data:
108
109         ax0 = 0xff00;
110         ax1 = 0x00ff;
111
112         ay0 = DM(Frame_tag);
113
114         call Bit_reversal;
115
116         ar = ax0 and ay0;            (mask off 8 thru 1 and set bits to zero)
117
118         sr1 = 0x001;
119
120         sr = sr or lshift ar by -7 (HI); (Shift data left 7 places
121                                         Bits 16 thru 9 now become bits
122                                         9 thru 2 and set bit 10 to one)
123
124         DM(IO,M0) = sr1;            (Transmit data)
125
126
127         ar = ax1 and ay0;            (mask off 16 thru 9 and set bits to zero)
128
129         sr1 = 0x001;
130
131         sr = sr or lshift ar by 1 (HI); (Shift data right 1 place
132                                         Bits 8 thru 1 now become bits 9
133                                         thru 2 set bit 10 to one)
134
135         DM(IO,M0) = sr1;            (Transmit data)
136
137
138
139         ay0 = DM(Elevation_error);
140
141         call Bit_reversal;
142
143         ar = ax0 and ay0;            (mask off 8 thru 1 and set bits to zero)
144
145         sr1 = 0x001;
146
147         sr = sr or lshift ar by -7 (HI); (Shift data left 7 places

```

```

148                                     Bits 16 thru 9 now become bits
149                                     9 thru 2 and set bit 10 to one)
150
151     DM(IO,M0) = sr1;                 {Transmit data}
152
153
154
155     ar = ax1 and ay0;                {mask off 16 thru 9 and set bits to zero}
156
157     sr1 = 0x001;
158
159     sr = sr or lshift ar by 1 (HI);   {Shift data right 1 place   Bits 8 thru 1
160                                     now become bits 9 thru 2}
161
162     DM(IO,M0) = sr1;                 {Transmit data}
163
164
165
166     ay0 = DM(Azimuth_error);
167
168     call Bit_reversal;
169
170     ar = ax0 and ay0;                {mask off 8 thru 1 and set bits to zero}
171
172     sr1 = 0x001;
173
174     sr = sr or lshift ar by -7 (HI);  {Shift data left 7 places   Bits 16 thru 9
175                                     now become bits 9 thru 2}
176
177     DM(IO,M0) = sr1;                 {Transmit data}
178
179
180     ar = ax1 and ay0;                {mask off 16 thru 9 and set bits to zero}
181
182     sr1 = 0x001;
183
184     sr = sr or lshift ar by 1 (HI);   {Shift data right 1 place   Bits 8 thru 1
185                                     now become bits 9 thru 2}
186
187     DM(IO,M0) = sr1;                 {Transmit data}
188
189
190
191     ay0 = DM(Total_signal_strength);
192
193     call Bit_reversal;
194
195     ar = ax0 and ay0;                {mask off 8 thru 1 and set bits to zero}
196
197     sr1 = 0x001;
198
199     sr = sr or lshift ar by -7 (HI);  {Shift data left 7 places   Bits 16 thru 9
200                                     now become bits 9 thru 2}
201
202     DM(IO,M0) = sr1;                 {Transmit data}
203
204
205
206     ar = ax1 and ay0;                {mask off 16 thru 9 and set bits to zero}
207
208     sr1 = 0x001;
209
210     sr = sr or lshift ar by 1 (HI);   {Shift data right 1 place   Bits 8 thru 1
211                                     now become bits 9 thru 2}
212
213     DM(IO,M0) = sr1;                 {Transmit data}
214
215     rts;
216
217
218     {#####}
219     {Send data sends the next data word to the host PC after receiving
220     a Sport0 transmit interrupt}
221

```

```
222 .ENTRY Send_data;
223
224 Send_data:
225
226     ENA Sec_reg;                {Enable secondary registers}
227
228     ay1 = dm(Number_of_words_sent);    {Test to see if all data has been}
229     ax1 = dm(Words_to_be_transmitted); {sent}
230     ar = ax1 - ay1;
231
232     if gt jump Transmit;        {If not done with data transmission
233                                 jump to transmit else quit transmission }
234
235     call Reset_address_pointer;   {If all data words have
236                                   been sent reset control flags
237                                   and exit interrupt routine}
238     call Sync_Pulse_Enable;
239
240     DIS Sec_reg;                {Disable secondary register}
241     nop;
242     RTI;
243
244 Transmit:
245
246     ar = dm(I0,M0);             {Transmit next data word to host PC}
247     tx0 = ar;
248
249     ar = ay1 + 1;               {Increment counter of words that has been sent}
250     dm(Number_of_words_sent)= ar;
251
252     DIS Sec_reg;                {Disable secondary register}
253     nop;
254
255     RTI;
256
257 {#####}
258 {Set_up_uart_parameters is used to initialize}
259 .ENTRY Set_up_uart_parameters;
260
261 Set_up_uart_parameters:
262
263     ar = 0;
264     dm(Number_of_words_sent) = ar;
265
266     ar = 8;
267     dm(Words_to_be_transmitted) = ar;
268
269     RTS;
270
271 {#####}
272 {Set_up_test_data is a subroutine used to set up predefined data}
273
274 Set_up_test_data:
275
276     ar = 0x01e0;                {Set up "T" for tranmission}
277     {dm(Frame_tag) = ar;}       {Transmit first word to host PC}
278
279     ar = 0x5556;                {Set up "S" for tranmission}
280     DM(Elevation_error) = ar;   {Transmit first word to host PC}
281
282     ar = 0x5678;                {Set up "I" for tranmission}
283     Dm(Azimuth_error) = ar;     {Transmit first word to host PC}
284
285     ar = 0xabc;                 {Set up "U" for tranmission}
286     Dm(Total_signal_strength) = ar; {Transmit first word to host PC}
287
288     RTS;
289
290 {#####}
291 {Bit reversal routine will rotate bit order 1 to 16 to the order of 16 to 1}
292
293 Bit_reversal:
294
295
```

```
296
297     cntr = 16;
298     mr0 = 0x0001;
299     mr1 = 0x8000;
300     ar = 0;
301     ay1 = 0;
302     se = 1;
303
304     Do Reverse_bit Until CE;
305
306     af = mr0 and ay0;      {Test to see if bit is a one}
307
308     if ne ar = mr1 or ay1; {Add bit to af register if bit x was a one}
309
310     sr = lshift mr0 (hi), ay1 = ar; {Shift masking bit right by one}
311                                     {Store new reversed data in ay1}
312     mr0 = sr1;                  {Store new value of mask word}
313
314     sr = lshift mr1 by -1 (hi); {Shift bit used for setting one to left}
315
316     Reverse_bit: mr1 = sr1;
317
318     ay0 = ar; {Store data back into ay0 for next step of formatting}
319
320     rts;
321
322     .endmod;
```

1 COMMON /ATTBLK/ NACQS,TSMALLSCAN

1

COMMON /CIRCBK/ PI,TWOPI,HALFPI,RAD


```
1      COMMON /DISCBLK/ DISCDEGMAX,DISCVEL,DISCACC,DISCACCTM,DRATIO,  
2      1  ALSETDEG,NSTEPS,ISTEPMAX,MAXSLEWSTEPS,MAXTRACKSTEPS,DLDEGPARK,  
3      2  ALDEGPARK,ISTEPAPARK,ISTEPBPARK  
4      INTEGER*4 NSTEPS,ISTEPMAX,MAXSLEWSTEPS,MAXTRACKSTEPS  
5      INTEGER*4 ISTEPAPARK,ISTEPBPARK
```

1

COMMON /FOVBLK/ FOVDEG,BEAMW,ARML

1 COMMON /MISCBLK/ REALTIME,TEST,WFLAG,IRATE,TRKGAIN
2 LOGICAL REALTIME,TEST,WFLAG

1 COMMON /ORBITBLK/ EPHH,EPHL,PRDH,PRDL(6)

1 COMMON /SAVEBLK/ INTRACKSAVE,USAVE,VSAVE,ISTEPASAVE,ISTEPBSAVE
2 INTEGER*4 ISTEPASAVE,ISTEPBSAVE
3 LOGICAL INTRACKSAVE

```

1  C-----
2
3      SUBROUTINE ALDLDLAB(AL,DL,ICHL,ISOL,INITDIR,TCUR,DLA,DLB,
4      1      ISTEPA,ISTEPB,ALSTEP)
5
6  C      This routine transforms from disc & arm angles to disc driver angles.
7  C
8  C      Prototype Version 1.0      Avtec Systems, Inc.      August 1992
9  C
10 C      Inputs:
11 C
12 C          AL      = Arm angles (alpha)
13 C          DL      = Disc angles (delta)
14 C          ICHL    = Channel index
15 C          ISOL    = Selected alpha & delta solution (1 or 2)
16 C          INITDIR = Initialize disc direction flags
17 C          TCUR    = Current time
18 C          ISTEPMAX = Disc step at maximum rotation (DISCBLK)
19 C          WFLAG   = Write flag (MISCBLK)
20 C
21 C      Outputs:
22 C
23 C          DLA(ICHL) = Disc driver A angle (delta A)
24 C          DLB(ICHL) = Disc driver B angle (delta B)
25 C          ISTEPA(ICHL) = Disc A resolution step
26 C          ISTEPB(ICHL) = Disc B resolution step
27 C          ALSTEP(ICHL) = Arm angle corresponding to ISTEPA(ICHL) and ISTEPB(ICHL)
28 C          INITDIR(ICHL) = Initialize disc direction flag
29 C
30 C-----
31
32      IMPLICIT REAL*8 (A-H,O-Z)
33      REAL*8 AL(6,2),DL(6,2),DLA(6),DLB(6),ALSTEP(6)
34      INTEGER*4 ISTEPA(6),ISTEPB(6),ISTEPAOLD(6),ISTEPBOLD(6)
35      INTEGER*4 ISTEPNORP,JSTEPA,JSTEPB
36      INTEGER*2 INITDIR(6),ISIGNOLD(6)
37      LOGICAL REVERSE
38      INCLUDE 'CIRCBK.FCB'
39      INCLUDE 'DISCBLK.FCB'
40      INCLUDE 'MISCBLK.FCB'
41
42      DATA INIT /1/
43
44      IF(INIT.EQ.1) THEN
45          IF(DISCDEGMAX.LT.180 .OR. DISCDEGMAX.GT.270) STOP 115
46          ALSET = ALSETDEG / RAD
47          DISCSTEP = TWOPI / NSTEPS
48          ISTEPNORP = NSTEPS - ISTEPMAX
49          INIT=0
50      END IF
51
52      REVERSE = .FALSE.
53      JSTEPA = NINT(DL(ICHL,ISOL)/DISCSTEP)
54      DLBCONT = DL(ICHL,ISOL) - (AL(ICHL,ISOL) - ALSET) / DRATIO
55      JSTEPB = NINT(DLBCONT/DISCSTEP)
56
57  C-----
58 C      If disc direction is already initialized and both old disc steps could be
59 C      negative or positive, maintain the same sign.
60 C-----
61
62      IF(INITDIR(ICHL).EQ.0 .AND. IABS(ISTEPAOLD(ICHL)).GE.ISTEPNORP
63      1 .AND. IABS(ISTEPBOLD(ICHL)).GE.ISTEPNORP) THEN
64
65          IF(ISIGN(1,JSTEPA).NE.ISIGNOLD(ICHL)) THEN
66
67              JSTEPA = JSTEPA + ISIGNOLD(ICHL)*NSTEPS
68              JSTEPB = JSTEPB + ISIGNOLD(ICHL)*NSTEPS
69
70          END IF
71
72      END IF
73

```

```
74 C-----
75 C   If either disc step exceeds maximum value, reverse both disc directions.
76 C-----
77
78       IF(IABS(JSTEPA).GT.ISTEPMAX .OR. IABS(JSTEPB).GT.ISTEPMAX) THEN
79           REVERSE = .TRUE.
80           JSTEPA = JSTEPA - ISIGN(NSTEPS,JSTEPA)
81           JSTEPB = JSTEPB - ISIGN(NSTEPS,JSTEPB)
82       END IF
83
84       IF(IABS(JSTEPA).GT.ISTEPMAX.OR.IABS(JSTEPB).GT.ISTEPMAX) STOP 116
85
86       IF(INITDIR(ICHL).EQ.0 .AND. REVERSE) THEN
87           IF(WFLAG) WRITE(15,20)
88       20   FORMAT(/17X,'ROTATION LIMIT / DIRECTION REVERSED')
89           WRITE(20,21) TCUR,ICHL
90       21   FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',12,
91           1   ' ROTATION LIMIT / DIRECTION REVERSED')
92       END IF
93
94       ISTEPA(ICHL) = JSTEPA
95       ISTEPB(ICHL) = JSTEPB
96
97 C-----
98 C   Compute disc angles and arm angle.
99 C-----
100
101       DLA(ICHL) = JSTEPA * DISCSTEP
102       DLB(ICHL) = JSTEPB * DISCSTEP
103
104       ALSTEP(ICHL) = ALSET + DRATIO * (DLA(ICHL) - DLB(ICHL))
105
106       ISTEPAOLD(ICHL) = JSTEPA
107       ISTEPBOLD(ICHL) = JSTEPB
108       ISIGNOLD(ICHL) = ISIGN(1,JSTEPA)
109       INITDIR(ICHL) = 0
110
111       RETURN
112       END
```

```

1  C-----
2
3      SUBROUTINE ANGLEPR (AZS,ELS,AZAHEAD,ELAHEAD,DLATRU,
4      1      ALTRUE,ILEO,ICHL,INTRACK,U,V)
5
6  C   In simulation mode, this routine computes and returns the
7  C   Angle Processor data.
8  C
9  C   In real-time mode, this routine returns the Angle Processor data
10 C   previously saved.
11 C
12 C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
13 C
14 C   Inputs:
15 C
16 C       AZS      = Azimuths of LEO satellites from Scene Processor
17 C       ELS      = Elevations of LEO satellites from Scene Processor
18 C       AZAHEAD  = Azimuth point-ahead angles
19 C       ELAHEAD  = Elevation point-ahead angles
20 C       DLATRU   = True disc A angles (delta A)
21 C       ALTRUE   = True arm angles (alpha)
22 C       ILEO     = LEO satellite index
23 C       ICHL     = Channel index
24 C       REALTIME  = Real-time flag (MISCBLK)
25 C       INTRACKSAVE = In-track flag in real-time mode (SAVEBLK)
26 C       USAVE,VSAVE = Tracking detector errors in real-time mode (SAVEBLK)
27 C
28 C   Outputs:
29 C
30 C       INTRACK(ILEO) = In-track flag
31 C       U(ILEO),V(ILEO) = Tracking detector errors
32 C
33 C-----
34
35      IMPLICIT REAL*8 (A-H,O-Z)
36      REAL*8 AZS(6),ELS(6),AZAHEAD(6),ELAHEAD(6)
37      REAL*8 DLATRU(6),ALTRUE(6),U(6),V(6)
38      LOGICAL INTRACK(6)
39      INCLUDE 'CIRCBK.FCB'
40      INCLUDE 'FOVBLK.FCB'
41      INCLUDE 'MISCBLK.FCB'
42      INCLUDE 'SAVEBLK.FCB'
43
44      DATA RMSERR /1.6D-5/      ! Tracking detector 2-axis rms error (rad)
45      DATA RES /1.25D-6/      ! Resolution of u,v errors (rad)
46      DATA TRMIS /1D-5/      ! Transmit/receive misalignment (rad)
47      DATA INIT /1/
48
49  C-----
50  C   In real-time mode, return the Angle Processor data previously saved.
51  C-----
52
53      IF(REALTIME) THEN
54          INTRACK(ILEO) = INTRACKSAVE
55          U(ILEO) = USAVE
56          V(ILEO) = VSAVE
57          RETURN
58      END IF
59
60  C-----
61  C   In simulation mode, compute and return the Angle Processor data.
62  C-----
63
64      IF(INIT.EQ.1) THEN
65          TANHAF = DTAN(5D-1*FOVDEG/RAD)
66          HALFBEAMSQ = (5D-1*BEAMW)**2
67          SIGMA = RMSERR / DSQRT(2D0)
68          CALL RNDM(UNIF)
69          ORIENTMIS = TWOPI*UNIF
70          TRMISHALF = 5D-1*TRMIS
71          INIT = 0
72      END IF
73

```



```

74 C-----
75 C   Determine coordinates of arm pivot point.
76 C-----
77
78     XA = ARML * DCOS(HALFPI-DLATRUE(ICHL))
79     YA = ARML * DSIN(HALFPI-DLATRUE(ICHL))
80
81 C-----
82 C   Determine coordinates of center of optics.
83 C-----
84
85     COSAL = DCOS(ALTRUE(ICHL))
86     SINAL = DSIN(ALTRUE(ICHL))
87
88     AZOPT = DATAN (TANHAF * (XA + COSAL*YA - SINAL*XA))
89     ELOPT = DATAN (TANHAF * (YA - SINAL*YA - COSAL*XA))
90
91 C-----
92 C   Include transmit/receive misalignment.
93 C-----
94
95     ANGLMIS = ORIENTMIS - DLATRUE(ICHL) - ALTRUE(ICHL)
96     COSMIS = DCOS(ANGLMIS)
97     SINMIS = DSIN(ANGLMIS)
98
99     AZOPTT = AZOPT + COSMIS*TRMISHALF
100    ELOPTT = ELOPT + SINMIS*TRMISHALF
101
102    AZOPTR = AZOPT - COSMIS*TRMISHALF
103    ELOPTR = ELOPT - SINMIS*TRMISHALF
104
105 C-----
106 C   Determine coordinates of LEO when HEO beam arrives. Note: errors in
107 C   point-ahead angles caused by small attitude and ephemeris errors are
108 C   negligible.
109 C-----
110
111    AZLEO = AZS(ILEO) + AZAHEAD(ILEO)
112    ELLEO = ELS(ILEO) + ELAHEAD(ILEO)
113
114 C-----
115 C   Assume in track if LEO is within HEO beam.
116 C-----
117
118    IF ((AZLEO-AZOPTT)**2 + (ELLEO-ELOPTT)**2 .LT. HALFBEAMSQ) THEN
119
120        INTRACK(ILEO) = .TRUE.
121
122 C-----
123 C   Determine difference coordinates of center of received spot.
124 C-----
125
126    AZDIF = AZS(ILEO) - AZOPTR
127    ELDIF = ELS(ILEO) - ELOPTR
128
129 C-----
130 C   Determine rotation from azimuth,elevation to u,v.
131 C-----
132
133    ROT = - (HALFPI + DLATRUE(ICHL) + ALTRUE(ICHL))
134    COSROT = DCOS(ROT)
135    SINROT = DSIN(ROT)
136
137 C-----
138 C   Determine random u,v errors.
139 C-----
140
141    UMEAN = AZDIF*COSROT + ELDIF*SINROT
142    VMEAN = -AZDIF*SINROT + ELDIF*COSROT
143
144    UCONT = GAUSCL(UMEAN,SIGMA,3DO)
145    VCONT = GAUSCL(VMEAN,SIGMA,3DO)
146
147    U(ILEO) = NINT(UCONT/RES) * RES

```

```
148          V(ILEO) = NINT(VCONT/RES) * RES
149
150 C-----
151 C   Assume not in track if LEO is outside HEO beam.
152 C-----
153
154         ELSE
155
156         INTRACK(ILEO) = .FALSE.
157
158         END IF
159
160         RETURN
161
162         END
```

```
1      DOUBLE PRECISION FUNCTION ARKTNS(N,X,Y)
2  C      FOUR QUAD ARC TANGENT X=cos(ANGLE) - Y=sin(ANGLE)
3  C      N=180 GIVES PV BETWEEN -PI & +PI
4  C      N=360 GIVES PV BETWEEN 0 & +2PI
5      IMPLICIT REAL*8 (A-H,O-Z)
6      PI=3.14159265358979323800
7      TPI=2*PI
8      IF (X.NE.0.000) GO TO 10
9      IF (Y.GT.0.000) T=0.500*PI
10     IF (Y.LT.0.000) T=1.500*PI
11     IF (Y.EQ.0.000) T=0.000
12     GO TO 20
13     10 T=DATAN(Y/X)
14     IF (X.LT.0.000) T=T+PI
15     IF (T.LT.0.000) T=T+TPI
16     20 IF (N.EQ.360) GO TO 30
17     IF (T.GT.PI) T=T-TPI
18     30 ARKTNS=T
19     RETURN
20     END
21
```

```

1 C-----
2
3 SUBROUTINE ASGNMFM (HEONAV,LEONAV,ECLIPSE,AZS,ELS,ILEO,ICHL,
4 1 LEOCHL,ICHLLEO,TCUR)
5
6 C The Assignment Function performs the following:
7 C
8 C Assigns disc channels to LEO satellites.
9 C Determines azimuth, elevation and point-ahead angles.
10 C Selects one of two possible solutions for the disc angle.
11 C Sets the scan reset flag for acquisition.
12 C
13 C Prototype Version 1.0 Avtec Systems, Inc. August 1992
14 C
15 C Entry points:
16 C
17 C ASGNMFM Assignment Function
18 C
19 C ASGNMFNR Read the predicted Az & El of LEO satellites and select
20 C alpha & delta solutions (in real-time mode)
21 C
22 C Inputs (ENTRY ASGNMFM):
23 C
24 C HEONAV = Geocentric-equatorial coordinates of HEO satellite from
25 C Navigation Processor
26 C LEONAV = Geocentric-equatorial coordinates of LEO satellites from
27 C Navigation Processor
28 C ECLIPSE = Eclipse flags from Scene Processor
29 C AZS = Azimuths of LEO satellites from Scene Processor
30 C ELS = Elevations of LEO satellites from Scene Processor
31 C ILEO = LEO satellite index
32 C ICHL = Channel index
33 C LEOCHL = LEO channel assignments
34 C ICHLLEO = Channel LEO assignments
35 C TCUR = Current time
36 C IRATE = Tracking rate (MISCBLK)
37 C REALTIME = Real-time flag (MISCBLK)
38 C WFLAG = Write flag (MISCBLK)
39 C
40 C Inputs (ENTRY ASGNMFNR):
41 C
42 C NLEOS = Number of LEO satellites
43 C NCHANNELS = Number of disc channels
44 C ICHLPRSEQ = Channel priority sequence
45 C
46 C-----
47
48 IMPLICIT REAL*8 (A-H,O-Z)
49 REAL*8 HEOSON(6),LEOSON(6,6)
50 REAL*8 HEONAV(6),LEONAV(6,6),HEONAVSOON(6),LEONAVSOON(6,6)
51 REAL*8 AZS(6),ELS(6),AZNAV(6),ELNAV(6)
52 REAL*8 AZAHEAD(6),ELAHEAD(6)
53 REAL*8 ALNAV(6,2),DLNAV(6,2)
54 INTEGER*2 LEOCHL(6),ICHLLEO(6),ICHLPRSEQ(6),IALDLSOL(6),INITDIR(6)
55 LOGICAL ECLIPSE(6),ECLPNAV(6),ECLPNAVSOON(6),INTRACK(6),SCANRES(6)
56 LOGICAL DONE
57 CHARACTER MODE(6)*13,DASH(72)*1
58 INCLUDE 'CIRCBK.FCB'
59 INCLUDE 'DISCBK.FCB'
60 INCLUDE 'MISCBLK.FCB'
61
62 DATA MODE /6*'PARK'/ ! Channel modes
63 DATA INTRACK /6*.FALSE./ ! In-track flags
64 DATA INITDIR /6*1/ ! Initialize disc direction flags
65 DATA DASH /72*'-'/
66 DATA INIT /1/
67
68 IF (INIT.EQ.1) THEN
69 TSLEWMAX = 2*DISCDEGMAX/DISCVEL
70 IF (WFLAG) WRITE(14,20) DASH
71 20 FORMAT(// ' ASSIGNMENT FUNCTION' // ' ',72A1)
72 INIT = 0
73 END IF

```

```

74
75
76     IF(ILEO.NE.0) THEN
77
78 C-----
79 C   If   (a) LEO is not currently assigned a channel, and
80 C       (b) Eclipse soon flag from Navigation Processor data is false
81 C
82 C   then (1) Assign the highest priority free channel, and
83 C       (2) Set mode to 'slew to acq', and
84 C       (3) Set scan reset flag to true.
85 C-----
86
87     IF(LEOCHL(ILEO).EQ.0) THEN
88
89         IF(.NOT.REALTIME) THEN
90             TSOON = TCUR + TSLEWMAX
91             CALL QVAL(TSOON,ILEO,HEOSOOON,LEOSOOON)
92             CALL NAVIGPR2(HEOSOOON,LEOSOOON,ILEO,TSOON,HEONAVSOON,
93 1             LEONAVSOON)
94             CALL XYZAZEL(HEONAVSOON,LEONAVSOON,ILEO,'N','T',
95 1             ECLPNAVSOON,AZNAVT,ELNAVT,AZAHEAD,ELAHEAD)
96         END IF
97
98         IF(.NOT.ECLPNAVSOON(ILEO)) THEN
99
100             LEOCHL(ILEO) = ICHL
101             ICHLLEO(ICHL) = ILEO
102             INITDIR(ICHL) = 1
103             MODE(ICHL) = 'SLEW TO ACQ'
104             SCANRES(ILEO) = .TRUE.
105
106             IF(WFLAG) WRITE(14,40) TCUR,ICHL,ILEO
107 40          FORMAT(// ' TIME =',F12.3,' SEC'//BX,'CHANNEL',I2,
108 1             ' ASSIGNED TO LEO',I2)
109             WRITE(20,45) TCUR,ICHL,ILEO
110 45          FORMAT(// ' TIME =',F12.3,' SEC',8X,'CHANNEL',I2,
111 1             ' ASSIGNED TO LEO',I2)
112
113             IF(.NOT.REALTIME) CALL XYZAZEL(HEONAV,LEONAV,ILEO,'N',
114 1             'T',ECLPNAV,AZNAVT,ELNAVT,AZAHEAD,ELAHEAD)
115
116         END IF
117
118 C-----
119 C   If   (a) LEO is currently assigned a channel, and
120 C       (b) Mode is not 'slew to acq', and
121 C       (c) Eclipse flag from Navigation Processor data is true, and
122 C       (d) In-track flag is false
123 C
124 C   then (1) Release the channel, and
125 C       (2) Set mode to 'park'.
126 C-----
127
128     ELSE
129
130         IF(.NOT.REALTIME) CALL XYZAZEL(HEONAV,LEONAV,ILEO,'N',
131 1             'T',ECLPNAV,AZNAVT,ELNAVT,AZAHEAD,ELAHEAD)
132
133         IF(MODE(ICHL).NE.'SLEW TO ACQ' .AND. ECLPNAV(ILEO) .AND.
134 1             .NOT.INTRACK(ILEO)) THEN
135
136             IF(WFLAG) WRITE(14,50) TCUR,ICHL,ILEO
137 50          FORMAT(// ' TIME =',F12.3,' SEC'//BX,'CHANNEL',I2,
138 1             ' RELEASED FROM LEO',I2)
139             WRITE(20,55) TCUR,ICHL,ILEO
140 55          FORMAT(// ' TIME =',F12.3,' SEC',8X,'CHANNEL',I2,
141 1             ' RELEASED FROM LEO',I2)
142
143             LEOCHL(ILEO) = 0
144             ICHLLEO(ICHL) = 0
145             INITDIR(ICHL) = 1
146             MODE(ICHL) = 'PARK'
147

```

```

148             END IF
149
150             END IF
151
152 C-----
153 C   If LEO is currently assigned a channel, convert from azimuth and elevation
154 C   to alpha and delta and select solution 1 (in simulation mode).
155 C-----
156
157             IF(LEOCHL(ILEO).NE.0) THEN
158
159                 IF(.NOT.REALTIME) THEN
160                     CALL AZELALDL(AZNAVT,ELNAVT,ILEO,ICHL,12,ALNAV,DLNAV)
161                     IALDLSOL(ICHL) = 1
162                 END IF
163
164                 IF(WFLAG) THEN
165
166                     ISOL = IALDLSOL(ICHL)
167
168                     WRITE(14,60) TCUR, ICHL, ILEO, AZNAVT(ILEO)*RAD,
169 1                     ELNAVT(ILEO)*RAD, DLNAV(ICHL,ISOL)*RAD,
170 2                     ALNAV(ICHL,ISOL)*RAD, ISOL, AZAHEAD(ILEO)*106,
171 3                     ELAHEAD(ILEO)*106
172 60             FORMAT(// ' TIME =',F12.3,' SEC',5X,'CHL',12,4X,'LEO',12//
173 1                 8X,'AZ NAV =',F11.5,7X,'EL NAV =',F10.5,' (DEG)'/8X,
174 2                 'DELTA =',F11.5,7X,'ALPHA =',F10.5,' (DEG)',9X,
175 3                 'SOL',12/8X,'AZ AHD =',F7.1,11X,'EL AHD =',F6.1,6X,
176 4                 '(MICRORAD)')
177
178                 END IF
179
180             END IF
181
182             END IF
183
184 C-----
185 C   Proceed to Control Function.
186 C-----
187
188             CALL CNTRLFN(ECLPNAV,AZNAVT,ELNAVT,IALDLSOL,AZAHEAD,ELAHEAD,
189 1                     ECLIPSE,MODE,INITDIR,INTRACK,SCANRES,ICHL,ILEO,
190 2                     AZS,ELS,TCUR)
191
192             RETURN
193
194
195 C-----
196 C   ENTRY ASGNMFNR(NLEOS,NCHANNELS,ICHLPRSEQ)
197
198
199 C   Read the predicted Az & El of LEO satellites and select alpha & delta
200 C   solutions (in real-time mode).
201 C-----
202
203             OPEN(2,FILE='AZEL.DAT',STATUS='OLD')
204
205             DO I=1,MIN(NLEOS,NCHANNELS)
206
207                 READ(2,*) AZPRED
208                 READ(2,*) ELPRED
209                 AZNAVT(I) = AZPRED/RAD
210                 ELNAVT(I) = ELPRED/RAD
211                 ECLPNAV(I) = .FALSE.
212                 ECLPNAVSOON(I) = .FALSE.
213                 AZAHEAD(I) = 0
214                 ELAHEAD(I) = 0
215
216                 J = ICHLPRSEQ(I)
217
218                 CALL AZELALDL(AZNAVT,ELNAVT,I,J,12,ALNAV,DLNAV)
219
220                 WRITE(*,90) I, AZPRED, ELPRED, DLNAV(J,1)*RAD, ALNAV(J,1)*RAD,
221 1                 DLNAV(J,2)*RAD, ALNAV(J,2)*RAD

```

```
222      90      FORMAT(/' LEO',I2,7X,'AZ PRED =',F11.5,6X,'EL PRED =',F11.5,
223      1      ' (DEG)'/13X,'DELTA 1 =',F11.5,6X,'ALPHA 1 =',F11.5,
224      2      ' (DEG)'/13X,'DELTA 2 =',F11.5,6X,'ALPHA 2 =',F11.5,
225      3      ' (DEG)')
226
227      DONE=.FALSE.
228      DO WHILE (.NOT.DONE)
229      91      WRITE(*,91)
230      91      FORMAT(/' Select solution (1 or 2)')
231      READ(*,*) IALDLSOL(J)
232      IF(IALDLSOL(J).EQ.1 .OR. IALDLSOL(J).EQ.2) DONE = .TRUE.
233      END DO
234
235      END DO
236
237      RETURN
238      END
```

```

1  C-----
2
3      SUBROUTINE AZELALDL(AZ,EL,ILEO,ICHL,ISOL,AL,DL)
4
5  C   This routine transforms FOV azimuth & elevation to
6  C   arm angle alpha and disc angle delta.
7  C
8  C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
9  C
10 C   Inputs:
11 C
12 C       AZ  = Azimuths of LEO satellites
13 C       EL  = Elevations of LEO satellites
14 C       ILEO = LEO satellite index
15 C       ICHL = Channel index
16 C
17 C       ISOL = 1 for solution #1
18 C              2 for solution #2
19 C              12 for both solutions
20 C
21 C   Outputs:
22 C
23 C       AL(ICHL,ISOL) = Arm angles alpha
24 C       DL(ICHL,ISOL) = Disc angles delta
25 C
26 C-----
27
28      IMPLICIT REAL*8 (A-H,O-Z)
29      REAL*8 AZ(6),EL(6),AL(6,2),DL(6,2)
30      INCLUDE 'CIRCBK.FCB'
31      INCLUDE 'FOVBLK.FCB'
32
33      DATA INIT /1/
34
35      IF(INIT.EQ.1) THEN
36          THREEHALFPI = 3*HALFPI
37          TANHALF = DTAN(FOVDEG*HALFPI/180)
38          IF(ARML.LE.1) STOP 111
39          INIT=0
40      END IF
41
42      XF = DTAN(AZ(ILEO)) / TANHALF
43      YF = DTAN(EL(ILEO)) / TANHALF
44
45      RF = DSQRT(XF**2 + YF**2)
46      IF(RF.EQ.0) STOP 112
47      IF(RF.GT.1) STOP 112
48
49      THETA = DATAN2(YF,XF)
50      PHI = DACOS(RF/(2*ARML))
51
52      IF(ISOL.EQ.1 .OR. ISOL.EQ.12) THEN
53          DELTA = HALFPI - THETA + PHI
54          IF(DELTA.GT.PI) DELTA = DELTA - TWOPI
55          DL(ICHL,1) = DELTA
56          AL(ICHL,1) = THREEHALFPI - 2*PHI
57      END IF
58
59      IF(ISOL.EQ.2 .OR. ISOL.EQ.12) THEN
60          DELTA = HALFPI - THETA - PHI
61          IF(DELTA.GT.PI) DELTA = DELTA - TWOPI
62          DL(ICHL,2) = DELTA
63          AL(ICHL,2) = 2*PHI - HALFPI
64      END IF
65
66      RETURN
67      END

```



```
1      BLOCK DATA
2
3      IMPLICIT REAL*8 (A-H,O-Z)
4
5      INCLUDE 'ATTBLK.FCB'
6
7      INCLUDE 'CIRCBLK.FCB'
8
9      INCLUDE 'DISCBLK.FCB'
10
11     INCLUDE 'FOVBLK.FCB'
12
13     INCLUDE 'MISCBLK.FCB'
14
15     INCLUDE 'ORBITBLK.FCB'
16
17     INCLUDE 'SAVEBLK.FCB'
18
19     DATA NACQS /0/           ! Number of acquisitions
20     DATA TSMALLSCAN /1030/   ! Earliest time for small scan pattern (sec)
21
22     DATA PI /3.14159265358979300/
23     DATA TWOPI /6.28318530717958600/
24     DATA HALFPI /1.57079632679489700/
25     DATA RAD /5.72957795130823201/
26
27     DATA DISCDEGMAX /2.402/   ! Maximum disc rotation (deg)
28     DATA DISCVEL /7.700/      ! Disc slew velocity (deg/sec)
29     DATA DISCACC /2.67403/    ! Disc acceleration (deg/sec**2)
30     DATA DISCACCTM /2.880-3/  ! Disc acceleration time (sec)
31     DATA DRATIO /2.729400/    ! Ratio of disc diameters
32     DATA ALSETDEG /901/       ! AL = 90deg when discs A & B are at step 0
33     DATA NSTEPS /708000/      ! Number of disc steps per revolution
34     DATA DLDEGPARK /000/      ! Park at delta = 0 deg
35     DATA ALDEGPARK /3500/     ! Park at alpha = 35 deg
36     DATA ISTEPAPARK /0/       ! Park at delta = 0 deg
37     DATA ISTEPBPARK /39630/   ! Park at alpha = 35 deg
38
39     DATA FOVDEG /2.201/       ! Field of view (deg)
40     DATA BEAMW /.50-4/        ! HEO beamwidth (rad)
41     DATA ARML /1.39600/       ! Arm pivot length relative to radius of FOV
42
43     DATA TEST /.FALSE./       ! Real-time test flag
44
45     DATA EPHH /20-1/          ! Ephemeris uncertainty for HEO orbit (km)
46     DATA EPHL /50-1/          ! Ephemeris uncertainty for LEO orbits (km)
47
48     END
```

```

1  C-----
2
3      SUBROUTINE CNTRLFN(ECLPNAV,AZNAVT,ELNAVT,IALDLSOL,AZAHEAD,ELAHEAD,
4      1      ECLIPSE,MODE,INITDIR,INTRACK,SCANRES,ICHL,ILEO,
5      2      AZS,ELS,TCUR)
6
7  C   The Control Function consists of:
8  C
9  C   Offset Processor: Error Processor, Offset Generator, and Scan Generator
10 C
11 C   Disc & Arm Controller: Determines angles for Disc Drivers
12 C
13 C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
14 C
15 C   Inputs:
16 C
17 C   ECLPNAV = Eclipse flags from Navigation Processor data
18 C   AZNAVT  = Azimuths of LEO satellites from Navigation Processor data
19 C   ELNAVT  = Elevations of LEO satellites from Navigation Processor data
20 C   IALDLSOL = Selected alpha & delta solutions (1 or 2)
21 C   AZAHEAD = Azimuth point-ahead angles
22 C   ELAHEAD = Elevation point-ahead angles
23 C   ECLIPSE = Eclipse flags from Scene Processor
24 C   MODE    = Channel modes
25 C   INITDIR  = Initialize disc direction flags
26 C   INTRACK  = In-track flags
27 C   SCANRES  = Scan reset flags
28 C   ICHL     = Channel index
29 C   ILEO     = LEO satellite index
30 C   AZS      = Azimuths of LEO satellites from Scene Processor
31 C   ELS      = Elevations of LEO satellites from Scene Processor
32 C   TCUR     = Current time
33 C   IRATE    = Tracking rate (MISCBLK)
34 C   REALTIME = Real-time flag (MISCBLK)
35 C   TRKGAIN  = Tracking gain (MISCBLK)
36 C   WFLAG    = Write flag (MISCBLK)
37 C   NACQS    = Number of acquisitions (ATTBLK)
38 C
39 C   Outputs:
40 C
41 C   MODE(ICHL) = Channel mode
42 C   INTRACK(ILEO) = In-track flag
43 C   NACQS      = Number of acquisitions (ATTBLK)
44 C   ISTEPASAVE = Disc A resolution step in real-time mode (SAVEBLK)
45 C   ISTEPBSAVE = Disc B resolution step in real-time mode (SAVEBLK)
46 C
47 C-----
48
49      IMPLICIT REAL*8 (A-H,O-Z)
50      REAL*8 AZNAVT(6),ELNAVT(6),AZAHEAD(6),ELAHEAD(6),AZS(6),ELS(6)
51      REAL*8 AZSCAN(6),ELSCAN(6),ACQDWEND(6)
52      REAL*8 AZSUM(6),ELSUM(6),AZERR(6),ELERR(6),U(6),V(6)
53      REAL*8 AZOFFSET(6),ELOFFSET(6),AZOFFSETP(6),ELOFFSETP(6)
54      REAL*8 AL(6,2),DL(6,2),ALSTEP(6)
55      REAL*8 DLA(6),DLB(6),DLATRUE(6),DLBTRUE(6),ALTRUE(6)
56      INTEGER*4 ISTEPA(6),ISTEPB(6),ISTEPAOLD(6),ISTEPBOLD(6)
57      INTEGER*4 ISTEPATRUE(6),ISTEPBTRUE(6)
58      INTEGER*4 INHSTEPS,INHCOUNT(6),INHCHECK(6)
59      INTEGER*2 IALDLSOL(6),INITCHL(6),INITDIR(6)
60      LOGICAL ECLPNAV(6),ECLIPSE(6),INTRACK(6),SCANRES(6)
61      CHARACTER DASH(79)*1,MODE(6)*13,INHMODE(6)*13,MODECHG*13
62      INCLUDE 'ATTBLK.FCB'
63      INCLUDE 'CIRCBLK.FCB'
64      INCLUDE 'DISCBLK.FCB'
65      INCLUDE 'FOVBLK.FCB'
66      INCLUDE 'MISCBLK.FCB'
67      INCLUDE 'SAVEBLK.FCB'
68
69      DATA ACQDWUR /10-1/      ! Duration of an acquisition dwell (sec)
70      DATA ISTEPTOLA /10/      ! Disc step tolerance for starting acquisition
71      DATA ISTEPTOLT /50/      ! Disc step tolerance for tracking
72      DATA FACTORINH /1.5D0/   ! Scan inhibit step factor
73      DATA DASH /79*'- '/

```

```

74      DATA INITCHL /6*1/
75      DATA INIT /1/
76
77      DATA DLATRUE /6*0D0/, DLBTRUE /6*0D0/, ALTRUE /6*0D0/
78      DATA DLA /6*0D0/, ALSTEP /6*0D0/
79      DATA ISTEPAOLD /6*0/, ISTEPBOLD /6*0/
80
81      IF(INIT.EQ.1) THEN
82
83          QRTBEAM = BEAMW/4
84          DLPARK = DLDEGPARK/RAD
85          ALPARK = ALDEGPARK/RAD
86          ACQDWDUR = ACQDWDUR - 1D-5
87          ISTEPMAX = NINT(NSTEPS*DISCDEGMAX/3.6D2)
88          MAXSLEWSTEPS = DISCVL/IRATE * NSTEPS/3.6D2
89
90          IF(DISCACCTM .GE. 5D-1/IRATE) THEN
91              MAXTRACKSTEPS = DISCACC/(2*IRATE)**2 * NSTEPS/3.6D2
92          ELSE
93              MAXTRACKSTEPS = (DISCACC*DISCACCTM**2 +
94 1              DISCACC*DISCACCTM*(1D0/IRATE-2*DISCACCTM)) * NSTEPS/3.6D2
95          END IF
96
97          INHSTEPS = MAXTRACKSTEPS / FACTORINH
98          IF(WFLAG) WRITE(15,20) DASH
99 20      FORMAT(// ' CONTROL FUNCTION'/// ' ',79A1)
100         INIT = 0
101
102     END IF
103
104     IF(TEST) THEN
105         MODE(ICHL) = 'TRACKING'
106         AZOFFSET(ILEO) = 0
107         ELOFFSET(ILEO) = 0
108     END IF
109
110     IF(WFLAG) WRITE(15,40) TCUR,ICHL,ILEO
111 40      FORMAT(// ' TIME =',F12.3,' SEC',5X,'CHL',12,4X,'LEO',12)
112
113     C-----
114     C   Get the true disc positions.
115     C-----
116
117     CALL DISCPOS(ISTEPAOLD,ISTEPBOLD,MODE,INITCHL,ICHL,ISTEPATRUE,
118 1         ISTEPBTRUE,DLATRUE,DLBTRUE,ALTRUE)
119
120     IF(WFLAG) WRITE(15,42) ISTEPATRUE(ICHL), ISTEPBTRUE(ICHL),
121 1     DLATRUE(ICHL)*RAD, DLBTRUE(ICHL)*RAD, ALTRUE(ICHL)*RAD
122 42      FORMAT(//3X,'TRUE POSITION'/17X,'STEP A =',I11,5X,'STEP B =',I11
123 1     /17X,'DELTA A =',F11.5,5X,'DELTA B =',F11.5,' (DEG)'
124 2     /42X,'ALPHA =',F11.5,' (DEG)')
125
126     C-----
127     C   If mode is 'slew to acq' and eclipse flag from Navigation Processor
128     C   data is false and true disc steps are close to previous commands, then
129     C   change mode to 'acquisition'.
130     C-----
131
132     IF(MODE(ICHL).EQ.'SLEW TO ACQ' .AND. .NOT.ECLPNAV(ILEO)) THEN
133
134         IF(INITDIR(ICHL).EQ.0 .AND.
135 1         IABS(ISTEPATRUE(ICHL)-ISTEPAOLD(ICHL)) .LE. ISTEPTOLA .AND.
136 2         IABS(ISTEPBTRUE(ICHL)-ISTEPBOLD(ICHL)) .LE. ISTEPTOLA) THEN
137
138             MODE(ICHL) = 'ACQUISITION'
139             WRITE(20,47) TCUR,ICHL
140 47      FORMAT(// ' TIME =',F12.3,' SEC',8X,'CHANNEL',12,
141 1         ' ACQUISITION STARTED')
142
143             ACQDWDEND(ILEO) = TCUR + ACQDWDUR
144         END IF
145     END IF
146
147     C-----

```

```

148 C If mode is 'slew to reacq' and true disc steps are close to previous
149 C commands, then change mode to 'reacquisition'.
150 C-----
151 IF(MODE(ICHL).EQ.'SLEW TO REACQ') THEN
152     IF(IABS(ISTEPATTRUE(ICHL)-ISTEPAOLD(ICHL)) .LE. ISTEPTOLA .AND.
153     1 IABS(ISTEPBTRUE(ICHL)-ISTEPBOLD(ICHL)) .LE. ISTEPTOLA) THEN
154         MODE(ICHL) = 'REACQUISITION'
155         WRITE(20,48) TCUR,ICHL
156         48 FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',12,
157             1 ' REACQUISITION STARTED')
158         ACQDWEND(ILEO) = TCUR + ACQDWDUR
159     END IF
160     END IF
161     IF(WFLAG) THEN
162         WRITE(15,55) MODE(ICHL)
163         55 FORMAT(/A16)
164         MODECHG = '
165     END IF
166 C-----
167 C If mode is 'acquisition', 'reacquisition' or 'tracking':
168 C-----
169 IF(MODE(ICHL).EQ.'ACQUISITION' .OR. MODE(ICHL).EQ.'REACQUISITION'
170 1 .OR. MODE(ICHL).EQ.'TRACKING') THEN
171 C-----
172 C If eclipsed, the LEO is not in track.
173 C-----
174 IF(ECLIPSE(ILEO)) THEN
175     INTRACK(ILEO) = .FALSE.
176     IF(WFLAG) WRITE(15,60)
177     60 FORMAT(17X,'NOT IN TRACK / ECLIPSE')
178     IF(MODE(ICHL).EQ.'TRACKING') WRITE(20,65) TCUR,ICHL
179     65 FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',12,
180         1 ' NOT IN TRACK / ECLIPSE')
181 ELSE
182 C-----
183 C If true disc steps are close to previous commands, then get information
184 C from the Angle Processors.
185 C-----
186 IF((IABS(ISTEPATTRUE(ICHL)-ISTEPAOLD(ICHL)) .LE. ISTEPTOLT .AND.
187 1 IABS(ISTEPBTRUE(ICHL)-ISTEPBOLD(ICHL)) .LE. ISTEPTOLT) .OR.
188 2 TEST) THEN
189     CALL ANGLEPR(AZS,ELS,AZAHEAD,ELAHEAD,DLATRUE,
190     1 ALTRUE,ILEO,ICHL,INTRACK,U,V)
191 C-----
192 C If in-track flag is true and mode is 'acquisition' or 'reacquisition'
193 C then change mode to 'tracking'.
194 C-----
195 IF(INTRACK(ILEO)) THEN
196     IF(MODE(ICHL).EQ.'ACQUISITION') THEN
197         NACQS = NACQS + 1
198         WRITE(20,70) TCUR,ICHL
199         70 FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',12,
200             1 ' ACQUISITION COMPLETED')
201     END IF
202

```

```

222
223      IF(MODE(ICHL).EQ.'REACQUISITION')WRITE(20,72)TCUR,ICHL
224      72      FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',I2,
225      1      ' REACQUISITION COMPLETED')
226
227      IF(MODE(ICHL).NE.'TRACKING') THEN
228          MODE(ICHL) = 'TRACKING'
229          MODECHG = 'TRACKING'
230      END IF
231
232  C-----
233  C   Error Processor transforms u,v errors from Angle Processors to
234  C   azimuth and elevation.
235  C-----
236
237      ROT = HALFPI + DLA(ICHL) + ALSTEP(ICHL)
238      COSROT = DCOS(ROT)
239      SINROT = DSIN(ROT)
240
241      AZERR(ILEO) = U(ILEO)*COSROT + V(ILEO)*SINROT +
242      1      AZAHEAD(ILEO)
243
244      ELERR(ILEO) = -U(ILEO)*SINROT + V(ILEO)*COSROT +
245      1      ELAHEAD(ILEO)
246
247      IF(WFLAG) WRITE(15,80) U(ILEO)*1D6, V(ILEO)*1D6,
248      1      AZERR(ILEO)*1D6, ELERR(ILEO)*1D6
249      80      FORMAT(17X,'U ERR =',F11.1,5X,'V ERR =',F11.1,
250      1      ' (MICRORAD)'/17X,'AZ ERR =',F11.1,5X,'EL ERR =',
251      2      F11.1,' (MICRORAD)')
252
253  C-----
254  C   Offset Generator updates azimuth and elevation offsets with
255  C   respective errors.
256  C-----
257
258      AZOFFSET(ILEO) = AZOFFSET(ILEO) + TRKGAIN*AZERR(ILEO)
259      ELOFFSET(ILEO) = ELOFFSET(ILEO) + TRKGAIN*ELERR(ILEO)
260
261      ELSE
262
263      IF(WFLAG) WRITE(15,90)
264      90      FORMAT(17X,'NOT IN TRACK / OUTSIDE HEO BEAM')
265
266      IF(MODE(ICHL).EQ.'TRACKING') THEN
267          IF(REALTIME) WRITE(20,95) TCUR,ICHL
268          IF(.NOT.REALTIME) WRITE(20,96) TCUR,ICHL
269      95      FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',I2,
270      1      ' NOT IN TRACK')
271      96      FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',I2,
272      1      ' NOT IN TRACK / OUTSIDE HEO BEAM')
273      END IF
274
275      END IF
276
277  C-----
278  C   If true disc steps are not close to previous commands, then change mode
279  C   to 'scan inhibit'.
280  C-----
281
282      ELSE
283
284          INHMODE(ICHL) = MODE(ICHL)
285          INHCOUNT(ICHL) = 1
286          INHCHECK(ICHL) = 0
287          MODE(ICHL) = 'SCAN INHIBIT'
288          IF(WFLAG) WRITE(15,55) MODE(ICHL)
289          WRITE(20,52) TCUR,ICHL
290      52      FORMAT(/' TIME =',F12.3,' SEC',8X,'CHANNEL',I2,
291      1      ' SCAN INHIBITED')
292
293      END IF
294
295      END IF

```

```
296
297 C-----
298 C   If mode is 'tracking' and in-track flag is false:
299 C       In real-time, set mode to 'slew to reacq' and set previous tracking
300 C       offsets to zero.
301 C       In simulation, set mode to 'reacquisition' and save previous tracking
302 C       offsets.
303 C       Set scan reset flag to true.
304 C-----
305
306         IF(MODE(ICHL).EQ.'TRACKING' .AND. .NOT.INTRACK(ILEO)) THEN
307
308 C Removed the following lines of code on 8-26-92 RJP
309
310 C         IF(REALTIME) THEN
311 C             MODE(ICHL) = 'SLEW TO REACQ'
312 C             MODECHG = 'SLEW TO REACQ'
313 C             AZOFFSETP(ILEO) = 0
314 C             ELOFFSETP(ILEO) = 0
315 C         ELSE
316 C             MODE(ICHL) = 'REACQUISITION'
317 C             IF(WFLAG) WRITE(15,55) MODE(ICHL)
318 C             WRITE(20,48) TCUR,ICHL
319 C             AZOFFSETP(ILEO) = AZOFFSET(ILEO)
320 C             ELOFFSETP(ILEO) = ELOFFSET(ILEO)
321 C         END IF
322
323         SCANRES(ILEO) = .TRUE.
324
325     END IF
326
327 END IF
328
329 C-----
330 C   If mode is 'park', set arm angle alpha and disc angle delta.
331 C-----
332
333         IF(MODE(ICHL).EQ.'PARK') THEN
334
335             ISOL = 1
336             AL(ICHL,ISOL) = ALPARK
337             DL(ICHL,ISOL) = DLPARK
338
339         ELSE
340
341 C-----
342 C   If mode is 'slew to acq' or 'acquisition' and a new scan position is
343 C   required, the Offset Generator uses the Scan Generator output.
344 C-----
345
346             IF((MODE(ICHL).EQ.'SLEW TO ACQ'.AND.SCANRES(ILEO)) .OR.
347 1             (MODE(ICHL).EQ.'ACQUISITION' .AND.
348 2             (SCANRES(ILEO).OR.TCUR.GT.ACQDWEND(ILEO)))) THEN
349
350                 CALL SCANGN(ILEO,ICHL,MODE,SCANRES,TCUR,AZOFFSET,ELOFFSET)
351                 IF(MODE(ICHL).EQ.'ACQUISITION')
352 1                     ACQDWEND(ILEO) = TCUR + ACQDWUR
353
354 C-----
355 C   If mode is 'slew to reacq' or 'reacquisition' and a new scan position is
356 C   required, the Offset Generator adds the Scan Generator output to the
357 C   previous tracking offsets. If mode is 'slew to reacq' these offsets were
358 C   set to zero.
359 C-----
360
361             ELSE IF((MODE(ICHL).EQ.'SLEW TO REACQ'.AND.SCANRES(ILEO)) .OR.
362 1             (MODE(ICHL).EQ.'REACQUISITION'.AND.
363 2             (SCANRES(ILEO).OR.TCUR.GT.ACQDWEND(ILEO)))) THEN
364
365                 CALL SCANGN(ILEO,ICHL,MODE,SCANRES,TCUR,AZSCAN,ELSCAN)
366                 AZOFFSET(ILEO) = AZOFFSETP(ILEO) + AZSCAN(ILEO)
367                 ELOFFSET(ILEO) = ELOFFSETP(ILEO) + ELSCAN(ILEO)
368                 IF(MODE(ICHL).EQ.'REACQUISITION')
369 1                     ACQDWEND(ILEO) = TCUR + ACQDWUR
```

```

370
371 C-----
372 C   If mode is 'scan inhibit', the Offset Generator performs a square test
373 C   pattern.
374 C-----
375
376       ELSE IF(MODE(ICHL).EQ.'SCAN INHIBIT') THEN
377
378           AZOFFSET(ILEO) = QRTBEAM * (-1)**(1+INHCOUNT(ICHL)/2)
379           ELOFFSET(ILEO) = QRTBEAM * (-1)**((1+INHCOUNT(ICHL))/2)
380
381       END IF
382
383       IF(WFLAG) WRITE(15,45) AZOFFSET(ILEO)*1D3, ELOFFSET(ILEO)*1D3
384 45   FORMAT(17X,'AZ OFFS =',F11.4,5X,'EL OFFS =',F11.4,
385 1     ' (MILLIRAD)')
386
387 C-----
388 C   Disc & Arm Controller adds azimuth and elevation from Navigation
389 C   Processor data and Offset Generator.
390 C-----
391
392       AZSUM(ILEO) = AZNAVT(ILEO) + AZOFFSET(ILEO)
393       ELSUM(ILEO) = ELNAVT(ILEO) + ELOFFSET(ILEO)
394
395 C-----
396 C   Azimuth and elevation are transformed to arm angle alpha and
397 C   disc angle delta.
398 C-----
399
400       ISOL = IALDLSOL(ICHL)
401
402       CALL AZELALDL(AZSUM,ELSUM,ILEO,ICHL,ISOL,AL,DL)
403
404       END IF
405
406 C-----
407 C   Arm angle alpha and disc angle delta are transformed to disc
408 C   driver A & B angles.
409 C-----
410
411       CALL ALDLDLAB(AL,DL,ICHL,ISOL,INITDIR,TCUR,DLA,DLB,
412 1     ISTEPA,ISTEPB,ALSTEP)
413
414 C-----
415 C   If mode is 'scan inhibit' and changes in disc step commands are small
416 C   enough for acquisition or reacquisition, change mode to 'slew to acq' or
417 C   'slew to reacq' depending on previous mode.
418 C-----
419
420       IF(MODE(ICHL).EQ.'SCAN INHIBIT') THEN
421
422           IF(INHCOUNT(ICHL).GE.2) THEN
423
424               IF(IABS(ISTEPA(ICHL)-ISTEPAOLD(ICHL)).LE.INHSTEPS .AND.
425 1             IABS(ISTEPB(ICHL)-ISTEPBOLD(ICHL)).LE.INHSTEPS) THEN
426
427                   IF(INHCHECK(ICHL).EQ.3) THEN
428
429                       IF(INHMODE(ICHL).EQ.'ACQUISITION') THEN
430                           MODE(ICHL) = 'SLEW TO ACQ'
431                           MODECHG = 'SLEW TO ACQ'
432                       ELSE
433                           MODE(ICHL) = 'SLEW TO REACQ'
434                           MODECHG = 'SLEW TO REACQ'
435                           AZOFFSETP(ILEO) = 0
436                           ELOFFSETP(ILEO) = 0
437                       END IF
438
439                       SCANRES(ILEO) = .TRUE.
440                       END IF
441
442                   INHCHECK(ICHL) = INHCHECK(ICHL) + 1
443

```

```
444         ELSE
445             INHCHECK(ICHL) = 0
446         END IF
447
448     END IF
449
450     INHCOUNT(ICHL) = INHCOUNT(ICHL) + 1
451
452 END IF
453
454 IF(WFLAG) WRITE(15,50) MODECHG, DL(ICHL,ISOL)*RAD,
455 1 AL(ICHL,ISOL)*RAD, ISOL, ISTEPA(ICHL), ISTEPB(ICHL),
456 2 DLA(ICHL)*RAD, DLB(ICHL)*RAD, ALSTEP(ICHL)*RAD
457
458 50 FORMAT(A16/17X,'DELTA  =',F11.5,5X,'ALPHA  =',F11.5,' (DEG)',
459 1 5X,'SOL',12/17X,'STEP A  =',I11,5X,'STEP B  =',I11/
460 2 17X,'DELTA A  =',F11.5,5X,'DELTA B  =',F11.5,' (DEG)'/
461 3 42X,'STEP AL  =',F11.5,' (DEG)')
462
463 -----
464 C-----
465 C   In real-time mode, save disc step commands.
466 C-----
467
468     IF(REALTIME) THEN
469         IF(TEST) THEN
470             ISTEPASAVE = ISTEPAPARK
471             ISTEPBSAVE = ISTEPBPARK
472         ELSE
473             ISTEPASAVE = ISTEPA(ICHL)
474             ISTEPBSAVE = ISTEPB(ICHL)
475         END IF
476     END IF
477
478     INITCHL(ICHL) = 0
479     ISTEPAOLD(ICHL) = ISTEPA(ICHL)
480     ISTEPBOLD(ICHL) = ISTEPB(ICHL)
481
482     RETURN
483     END
```



```
1      SUBROUTINE COPYVEC(U,N,V)
2      REAL*8 U(N),V(N)
3      DO I=1,N
4          V(I)=U(I)
5      END DO
6      RETURN
7      END
```

```
1      SUBROUTINE CROSS(U,V,W)
2      REAL*8 U(3),V(3),W(3)
3      W(1)=U(2)*V(3)-U(3)*V(2)
4      W(2)=U(3)*V(1)-U(1)*V(3)
5      W(3)=U(1)*V(2)-U(2)*V(1)
6      RETURN
7      END
```

```
1 C-----
2
3 SUBROUTINE DFQ(X,DX,DT)
4 C
5 C Differential equations for satellite state X.
6 C
7 C Courtesy of Bob Dasenbrock of NRL. Modified for OMA.
8 C
9 C-----
10
11 IMPLICIT REAL*8 (A-H,O-Z)
12 REAL*8 MU,MUR
13 DIMENSION X(6),DX(6)
14 DATA MU /-3.986008D5/
15
16 R=DSQRT(X(1)**2+X(2)**2+X(3)**2)
17 MUR=MU/(R**3)
18
19 DX(1)=X(4)*DT
20 DX(2)=X(5)*DT
21 DX(3)=X(6)*DT
22
23 DX(4)=MUR*X(1)*DT
24 DX(5)=MUR*X(2)*DT
25 DX(6)=MUR*X(3)*DT
26
27 RETURN
28 END
```

```

1  C-----
2      SUBROUTINE DISCPOS (ISTEPAOLD,ISTEPBOLD,MODE,INITCHL,ICHL,
3      1 ISTEPATRUE,ISTEPBTRUE,DLATRUE,DLBTRUE,ALTRUE)
4
5      In simulation mode, this routine computes and returns the
6      C true disc positions.
7      C
8      C In real-time mode, this routine returns the true disc positions
9      C previously saved.
10     C
11     C Prototype Version 1.0      Avtec Systems, Inc.      August 1992
12     C
13     C Inputs:
14     C
15     C     ISTEPAOLD      = Previous disc A step commands
16     C     ISTEPBOLD      = Previous disc B step commands
17     C     MODE          = Channel modes
18     C     INITCHL       = Channel initializations
19     C     ICHL          = Channel index
20     C     ISTEPMAX       = Disc step at maximum rotation (DISCBLK)
21     C     MAXSLEWSTEPS   = Max #steps in an update interval if slewing (DISCBLK)
22     C     MAXTRACKSTEPS  = Max #steps in an update interval if tracking (DISCBLK)
23     C     IRATE         = Tracking rate (MISCBLK)
24     C     REALTIME      = Real-time flag (MISCBLK)
25     C     ISTEPASAVE     = True disc A step in real-time mode (SAVEBLK)
26     C     ISTEPBSAVE     = True disc B step in real-time mode (SAVEBLK)
27     C
28     C Outputs:
29     C
30     C     ISTEPATRUE(ICHL) = True disc A step
31     C     ISTEPBTRUE(ICHL) = True disc B step
32     C     DLATRUE(ICHL)    = True disc A angle (delta A) in simulation mode
33     C     DLBTRUE(ICHL)    = True disc B angle (delta B) in simulation mode
34     C     ALTRUE(ICHL)     = True arm angle (alpha) in simulation mode
35     C
36     C-----
37     C
38     IMPLICIT REAL*8 (A-H,O-Z)
39     REAL*8 DLATRUE(6),DLBTRUE(6),ALTRUE(6),APHS(6),BPHS(6)
40     INTEGER*4 ISTEPAOLD(6),ISTEPBOLD(6),ISTEPATRUE(6),ISTEPBTRUE(6)
41     INTEGER*4 ISTEPATRUEOLD(6),ISTEPBTRUEOLD(6)
42     INTEGER*4 ISTEPDIFF,JSTEPATRUE,JSTEPBTRUE,MAXSTEPS
43     INTEGER*2 INITCHL(6)
44     CHARACTER*13 MODE(6)
45     INCLUDE 'CIRCBLK.FCB'
46     INCLUDE 'DISCBLK.FCB'
47     INCLUDE 'MISCBLK.FCB'
48     INCLUDE 'SAVEBLK.FCB'
49
50     DATA ACCUR /2D0/      ! Disc step accuracy
51     DATA F /4D0/         ! Period factor for accuracy model
52     DATA MAXSTEPERR /2/   ! Maximum random step error
53     DATA INIT /1/
54
55     C-----
56     C In real-time mode, return the true disc positions previously saved.
57     C-----
58     C
59     IF(REALTIME) THEN
60         ISTEPATRUE(ICHL) = ISTEPASAVE
61         ISTEPBTRUE(ICHL) = ISTEPBSAVE
62         RETURN
63     END IF
64
65     C-----
66     C In simulation mode, compute and return the true disc positions.
67     C-----
68     C
69     IF(INIT.EQ.1) THEN
70         DISCSTEP = TWOPI / NSTEPS
71         DLMAX = DISCSTEP * (ISTEPMAX + 4.999D-1)
72         ERRAMPL = ACCUR * DISCSTEP
73

```

```

74      ALSET = ALSETDEG / RAD
75      DO I=1,6
76          CALL RNDM(UNIF)
77          APHS(I) = TWOPI*UNIF
78          CALL RNDM(UNIF)
79          BPHS(I) = TWOPI*UNIF
80      END DO
81      INIT=0
82  END IF
83
84  C-----
85  C   Compute the true disc steps.  Simulation begins in park positions.
86  C-----
87
88      IF(INITCHL(ICHL).EQ.1) THEN
89
90          JSTEPATRUE = ISTEPAPARK
91          JSTEPBTRUE = ISTEPBPARK
92
93      ELSE
94
95          IF(MODE(ICHL).EQ.'ACQUISITION'.OR.MODE(ICHL).EQ.'REACQUISITION'
96      1      .OR.MODE(ICHL).EQ.'TRACKING') THEN
97              MAXSTEPS = MAXTRACKSTEPS
98          ELSE
99              MAXSTEPS = MAXSLEWSTEPS
100      END IF
101
102      ISTEPAOLD(ICHL) = ISTEPATRUEOLD(ICHL)
103      JSTEPATRUE = ISTEPATRUEOLD(ICHL) +
104      1      ISIGN(1,ISTEPDIFF) * MIN(MAXSTEPS,IABS(ISTEPDIFF))
105
106      ISTEPBOLD(ICHL) = ISTEPBTRUEOLD(ICHL)
107      JSTEPBTRUE = ISTEPBTRUEOLD(ICHL) +
108      1      ISIGN(1,ISTEPDIFF) * MIN(MAXSTEPS,IABS(ISTEPDIFF))
109  END IF
110
111  C-----
112  C   Include random disc step errors.
113  C-----
114
115      IF(INITCHL(ICHL).EQ.1 .OR.
116      1      (MODE(ICHL).EQ.'PARK' .AND. JSTEPATRUE.EQ.ISTEPAPARK)) THEN
117
118          DLRNDMA = DISCSTEP * ISTEPAPARK
119
120      ELSE
121
122          CALL RNDM(UNIF)
123          DLRNDMA = DISCSTEP * (JSTEPATRUE + (2*UNIF-1) * MAXSTEPERR)
124          IF(DABS(DLRNDMA).GT.DLMAX) DLRNDMA = DSIGN(DLMAX,DLRNDMA)
125          JSTEPATRUE = NINT(DLRNDMA/DISCSTEP)
126
127      END IF
128
129      IF(INITCHL(ICHL).EQ.1 .OR.
130      1      (MODE(ICHL).EQ.'PARK' .AND. JSTEPBTRUE.EQ.ISTEPBPARK)) THEN
131
132          DLRNDMB = DISCSTEP * ISTEPBPARK
133
134      ELSE
135
136          CALL RNDM(UNIF)
137          DLRNDMB = DISCSTEP * (JSTEPBTRUE + (2*UNIF-1) * MAXSTEPERR)
138          IF(DABS(DLRNDMB).GT.DLMAX) DLRNDMB = DSIGN(DLMAX,DLRNDMB)
139          JSTEPBTRUE = NINT(DLRNDMB/DISCSTEP)
140
141      END IF
142
143      ISTEPATRUE(ICHL) = JSTEPATRUE
144      ISTEPBTRUE(ICHL) = JSTEPBTRUE
145
146  C-----
147  C   Determine true disc and arm angles.

```

```
148 C-----
149
150      DLATRUE(ICHL) = DLRNDMA + ERRAMPL * DSIN(F*DLRNDMA + APHS(ICHL))
151      DLBTRUE(ICHL) = DLRNDMB + ERRAMPL * DSIN(F*DLRNDMB + BPHS(ICHL))
152
153      ALTRUE(ICHL) = ALSET + DRATIO * (DLATRUE(ICHL)-DLBTRUE(ICHL))
154
155 C-----
156 C   Save true disc steps.
157 C-----
158
159      ISTEPATRUEOLD(ICHL) = JSTEPATRUE
160      ISTEPBTRUEOLD(ICHL) = JSTEPBTRUE
161
162      RETURN
163      END
```

```
1      REAL*8 FUNCTION DOT(U,V)
2      REAL*8 U(3),V(3)
3      DOT=U(1)*V(1)+U(2)*V(2)+U(3)*V(3)
4      RETURN
5      END
```

```

1      SUBROUTINE ELSTAT (ELD,X)
2      C      ELD IN THE INPUT KEPLERIAN STATE IN KM AND DEGREES
3      C      X IS THE OUTPUT CARTESIAN STATE IN KILOMETERS
4      C      WHERE:
5      C          ELD(1)=SEMI-MAJOR AXIS (KM)
6      C          ELD(2)=ECCENTRICITY
7      C          ELD(3)=INCLINATION (DEG)
8      C          ELD(4)=LOA OF ASC NODE (DEG)
9      C          ELD(5)=ARG OF PERIGEE (DEG)
10     C          ELD(6)=MEAN ANOMALY (DEG)
11     C
12     C          X(1)=X (KM)
13     C          X(2)=Y (KM)
14     C          X(3)=Z (KM)
15     C          X(4)=XDOT (KM/S)
16     C          X(5)=YDOT (KM/S)
17     C          X(6)=ZDOT (KM/S)
18     C
19     C      IMPLICIT REAL*8 (A-H,O-Z)
20     C      DIMENSION X(6), ELD(6), A(3,2)
21     C      XMU=398600.800
22     C      PI=3.14159265358979323800
23     C      DTR=PI/180.000
24     C      SNI=DSIN(ELD(3)*DTR)
25     C      CNI=DCOS(ELD(3)*DTR)
26     C      SOM=DSIN(ELD(4)*DTR)
27     C      COM=DCOS(ELD(4)*DTR)
28     C      XM=DMOD(ELD(6),360.000)*DTR
29     C      ECC=ELD(2)
30     C      E=XKEP(ECC,XM,1.0D-14)
31     C      SINE=DSIN(E)
32     C      COSE=DCOS(E)
33     C      STA=SQRT(1.000-ECC**2)*SINE/(1.000-ECC*COSE)
34     C      CTA=(COSE-ECC)/(1.000-ECC*COSE)
35     C      TAA=ARCTNS(180,CTA,STA)
36     C      TBB=TAA+DTR*ELD(5)
37     C      CBA=DCOS(TBB)
38     C      SBA=DSIN(TBB)
39     C      A(1,1)=+COM*CBA-SOM*CNI*SBA
40     C      A(2,1)=+SOM*CBA+COM*CNI*SBA
41     C      A(3,1)=+SNI*SBA
42     C      A(1,2)=-COM*SBA-SOM*CNI*CBA
43     C      A(2,2)=-SOM*SBA+COM*CNI*CBA
44     C      A(3,2)=+SNI*CBA
45     C      P=ELD(1)*(1.000-ECC**2)
46     C      R=P/(1.000+ECC*CTA)
47     C      VR=ECC*STA*SQRT(XMU/P)
48     C      VT=SQRT(XMU*(2.000/R-1.000/ELD(1))-VR*VR)
49     C      DO 10 K=1,3
50     C          X(K)=R*A(K,1)
51     C      10 X(K+3)=VR*A(K,1)+VT*A(K,2)
52     C      RETURN
53     C      END
54

```



```
1 C-----
2
3     FUNCTION GAUSCL(XMEAN,SIGMA,CUTOFF)
4
5 C   GENERATES A RANDOM NUMBER FROM A CLIPPED GAUSSIAN DISTRIBUTION, WHERE
6 C   THE TAILS ARE CUT OFF.
7 C
8 C   INPUTS:
9 C       XMEAN = MEAN OF GAUSSIAN DISTRIBUTION
10 C       SIGMA = STANDARD DEVIATION OF GAUSSIAN DISTRIBUTION
11 C       CUTOFF = CUTOFF VALUE IN UNITS OF SIGMA
12 C
13 C   OUTPUT:
14 C       GAUSCL = RANDOM NUMBER FROM CLIPPED GAUSSIAN DISTRIBUTION
15 C
16 C-----
17
18     IMPLICIT REAL*8 (A-H,O-Z)
19     50 GAUSCL = GAUSS(0D0,1D0)
20     IF(DABS(GAUSCL) .GT. CUTOFF) GO TO 50
21     GAUSCL = XMEAN + GAUSCL*SIGMA
22     RETURN
23     END
```

```
1 C-----
2
3     FUNCTION GAUSS (XMEAN,VARIANCE)
4
5 C   GENERATES A RANDOM NUMBER FROM A GAUSSIAN DISTRIBUTION
6 C
7 C   INPUTS:
8 C     XMEAN  = MEAN OF GAUSSIAN DISTRIBUTION
9 C     VARIANCE = VARIANCE OF GAUSSIAN DISTRIBUTION
10 C
11 C   OUTPUT:
12 C     GAUSS = RANDOM NUMBER
13 C
14 C   REFERENCE:
15 C     L.R. RABINER AND B. GOLD, "THEORY AND APPLICATION OF DIGITAL SIGNAL
16 C     PROCESSING", PRENTICE-HALL, ENGLEWOOD CLIFFS, NJ, 1975, pp.570-571.
17 C
18 C-----
19
20     IMPLICIT REAL*8 (A-H,O-Z)
21
22     DATA TWOPI/6.2831853071795865D0/
23
24     CALL RNDM(X1)
25     CALL RNDM(X2)
26     Y = DSQRT(-2*VARIANCE*DLOG(X1))
27     GAUSS = XMEAN + Y*DCOS(TWOPI*X2)
28     RETURN
29     END
```

```
1 C-----
2
3 SUBROUTINE INTRFACE
4
5 C Interface between control software and executive program.
6 C
7 C Prototype Version 1.0 Avtec Systems, Inc. August 1992
8 C
9 C Entry points:
10 C
11 C INIT Initialization of control software
12 C POSITION Computation of new position commands for a disc channel
13 C
14 C Inputs:
15 C
16 C REALTIMEFLAG = Real-time flag (LOGICAL), argument in ENTRY INIT:
17 C
18 C True indicates real-time mode
19 C False indicates simulation mode
20 C
21 C In real-time mode:
22 C
23 C Input file name is INPUT.DAT
24 C Az-El file name is AZEL.DAT (see SUBROUTINE ASGNMFN)
25 C
26 C In simulation mode, input file name (CHARACTER*12) is entered from
27 C the keyboard
28 C
29 C The following inputs are contained in the input file:
30 C
31 C SIMHRS = Duration of simulation in hrs (REAL*8)
32 C (used only in simulation mode)
33 C
34 C NCHANNELS = Number of disc channels (INTEGER: 1-6)
35 C
36 C IRATE = Tracking rate in Hz (INTEGER)
37 C
38 C TRKGAIN = Tracking gain (REAL*8)
39 C
40 C WFLAG = Write flag for output from the Truth Generator, Scene
41 C Processor, Navigation Processor, Assignment Function
42 C and Control Function (LOGICAL, false in real-time mode)
43 C
44 C Output file names (CHARACTER*12) are entered from
45 C the keyboard
46 C
47 C HEL(J) = Classical orbital elements for HEO at time 0 (REAL*8):
48 C
49 C (1) = Semi-major axis (km)
50 C (2) = Eccentricity
51 C (3) = Inclination (deg)
52 C (4) = Longitude of ascending node (deg)
53 C (5) = Argument of perigee (deg)
54 C (6) = Mean anomaly (deg)
55 C
56 C NLEOS = Number of LEO satellites (INTEGER: 1-6)
57 C
58 C LEL(J,I) = Classical orbital elements for LEO I at time 0 (REAL*8)
59 C
60 C Outputs:
61 C
62 C Event Log
63 C
64 C Truth Generator (if WFLAG is true)
65 C Scene Processor (if WFLAG is true)
66 C Navigation Processor (if WFLAG is true)
67 C Assignment Function (if WFLAG is true)
68 C Control Function (if WFLAG is true)
69 C
70 C In real-time mode, Event Log file name is EVENT.DAT
71 C
72 C In simulation mode, output file names (CHARACTER*12) are entered from
73 C the keyboard
```

```

74 C
75 C-----
76
77     IMPLICIT REAL*8 (A-H,O-Z)
78     REAL*8 U [FAR]
79     REAL*8 V [FAR]
80     REAL*8 HEL(6),LEL(6,6),HEO(6),LEO(6,6),MU
81     INTEGER*4 ISTEPA [FAR]
82     INTEGER*4 ISTEPB [FAR]
83     INTEGER*2 ICHANNEL [FAR]
84     INTEGER*2 ICHLORDER(6),ICHLPRSEQ(6),ICHLLEO(6),LEOCHL(6)
85     CHARACTER*12 FILENAME
86     LOGICAL REALTIMEFLAG [FAR]
87     LOGICAL INTRACK [FAR]
88     LOGICAL DONE
89     INCLUDE 'CIRCBK.FCB'
90     INCLUDE 'MISCBK.FCB'
91     INCLUDE 'ORBITBK.FCB'
92     INCLUDE 'SAVEBK.FCB'
93
94     DATA ICHLORDER /1,2,3,4,5,6/      ! Channel command order
95     DATA IORDER /0/                    ! Channel order index
96     DATA ICHLPRSEQ /1,2,3,4,5,6/      ! Channel priority sequence
97     DATA ILEOSEQ /0/                   ! LEO sequence
98     DATA LEOCHL /6*0/                  ! LEO channel assignments
99     DATA ICHLLEO /6*0/                 ! Channel LEO assignments
100    DATA INITP /1/                      ! Initialization of ENTRY POSITION
101
102    DATA RE /6.378145D3/                 ! Earth radius (km)
103    DATA MU /3.986008D5/                 ! Gravitational parameter
104
105 C-----
106     ENTRY INIT-REALTIMEFLAG)
107
108 C Initialization of control software.
109 C-----
110
111     REALTIME = REALTIMEFLAG
112
113     IF-REALTIME) THEN
114         OPEN(1,FILE='INPUT.DAT',STATUS='OLD')
115     ELSE
116         WRITE(*,1)
117         1   FORMAT(' Enter input file name (e.g., Up to 8 characters.DAT)')
118         READ(*,2) FILENAME
119         2   FORMAT(A)
120         OPEN(1,FILE=FILENAME,STATUS='OLD')
121     END IF
122
123 C-----
124 C Read input variables.
125 C-----
126
127     READ(1,*) SIMHRS                      ! Used only in simulation mode
128     READ(1,*) NCHANNELS
129     READ(1,*) IRATE
130     READ(1,*) TRKGAIN
131     READ(1,*) WFLAG
132     IF-REALTIME) WFLAG = .FALSE.
133
134     IF(NCHANNELS.LT.1 .OR. NCHANNELS.GT.6) STOP 151
135     IF(IRATE.LT.1) STOP 152
136     IF(TRKGAIN.LE.0) STOP 153
137
138 C-----
139 C Read classical orbital elements for HEO and up to 6 LEO satellites
140 C at time 0 and transform to geocentric-equatorial coordinates and
141 C velocity components.
142 C-----
143
144     DO J=1,6
145         READ(1,*) HEL(J)
146     END DO
147

```

```
148      IF(HEL(1).LE.RE) STOP 161
149      IF(HEL(2).NE.0) STOP 162
150      IF(HEL(3).LT.0 .OR. HEL(3).GT.180) STOP 163
151
152      DO J=4,6
153          IF(DABS(HEL(J)).GE.360) STOP 164
154      END DO
155
156      TEMP = TWOPI / DSQRT(MU)
157      PRDH = TEMP * HEL(1)**1.5D0
158      CALL ELSTAT(HEL,HEO)
159
160      READ(1,*) NLEOS
161      IF(NLEOS.LT.1 .OR. NLEOS.GT.6) STOP 154
162
163      DO I=1,NLEOS
164
165          DO J=1,6
166              READ(1,*) LEL(J,I)
167          END DO
168
169          IF(LEL(1,I).LE.RE) STOP 161
170          IF(LEL(2,I).LT.0 .OR. LEL(2,I).GE.1) STOP 162
171          IF(LEL(3,I).LT.0 .OR. LEL(3,I).GT.180) STOP 163
172
173          DO J=4,6
174              IF(DABS(LEL(J,I)).GE.360) STOP 164
175          END DO
176
177          PRDL(I) = TEMP * LEL(1,I)**1.5D0
178          CALL ELSTAT(LEL(1,I),LEO(1,I))
179
180      END DO
181
182      C-----
183      C   In real-time mode, read the predicted Az & El of LEO satellites and
184      C   select alpha & delta solutions.
185      C-----
186
187      IF(REALTIME) CALL ASGNMFNR(NLEOS,NCHANNELS,ICHLRSEQ)
188
189      C-----
190      C   Read the names of output files.
191      C-----
192
193      IF(WFLAG) THEN
194
195          WRITE(*,41)
196          41  FORMAT(' Enter file name for Truth Generator output',
197                1  ' (e.g., Up to 8 characters.DAT)')
198          READ(*,2) FILENAME
199          OPEN(11,FILE=FILENAME,STATUS='NEW')
200
201          WRITE(*,42)
202          42  FORMAT(' Enter file name for Scene Processor output',
203                1  ' (e.g., Up to 8 characters.DAT)')
204          READ(*,2) FILENAME
205          OPEN(12,FILE=FILENAME,STATUS='NEW')
206
207          WRITE(*,43)
208          43  FORMAT(' Enter file name for Navigation Processor output',
209                1  ' (e.g., Up to 8 characters.DAT)')
210          READ(*,2) FILENAME
211          OPEN(13,FILE=FILENAME,STATUS='NEW')
212
213          WRITE(*,44)
214          44  FORMAT(' Enter file name for Assignment Function output',
215                1  ' (e.g., Up to 8 characters.DAT)')
216          READ(*,2) FILENAME
217          OPEN(14,FILE=FILENAME,STATUS='NEW')
218
219          WRITE(*,45)
220          45  FORMAT(' Enter file name for Control Function output',
221                1  ' (e.g., Up to 8 characters.DAT)')
```

```

222         READ(*,2) FILENAME
223         OPEN(15,FILE=FILENAME,STATUS='NEW')
224
225     END IF
226
227     IF(REALTIME) THEN
228         OPEN(20,FILE='EVENT.DAT',STATUS='UNKNOWN')
229     ELSE
230         WRITE(*,46)
231         46   FORMAT(' Enter file name for Event Log output',
232             1   ' (e.g., Up to 8 characters.DAT)')
233         READ(*,2) FILENAME
234         OPEN(20,FILE=FILENAME,STATUS='NEW')
235     END IF
236
237     WRITE(20,60)
238     60   FORMAT(// ' EVENT LOG'/' -----')
239
240     TCUR = 0
241     TSTOP = SIMHRS*3.603
242     DT = 1D0/(IRATE*NCHANNELS)
243
244     RETURN
245
246
247 C-----
248     ENTRY POSITION (ICHANNEL,INTRACK,U,V,ISTEPA,ISTEPB)
249
250 C   Computation of new position commands for a disc channel.
251 C
252 C   Inputs (real-time mode only):
253 C
254 C       ICHANNEL = Channel index
255 C       INTRACK  = In-track flag
256 C       U,V      = Tracking detector errors (rad)
257 C       ISTEPA   = Current disc A resolution step
258 C       ISTEPB   = Current disc B resolution step
259 C
260 C   Outputs (real-time mode only):
261 C
262 C       ISTEPA   = New command for disc A resolution step
263 C       ISTEPB   = New command for disc B resolution step
264 C
265 C-----
266
267 C-----
268 C   In real-time mode, save Angle Processor inputs and current disc step
269 C   positions. Also update current time and determine LEO satellite index.
270 C   Note: all channel assignments are performed in SUBROUTINE ASGNMFM.
271 C-----
272
273     IF(REALTIME) THEN
274
275         IF(U.GT.1D1 .OR. V.GT.1D1) THEN
276             TEST = .TRUE.
277             INTRACK = .TRUE.
278             U = 0
279             V = 0
280         ELSE
281             TEST = .FALSE.
282         END IF
283
284         INTRACKSAVE = INTRACK
285         USAVE = U
286         VSAVE = V
287         ISTEPASAVE = ISTEPA
288         ISTEPBSAVE = ISTEPB
289
290         ICHL = ICHANNEL
291         IORDER = 0
292         DONE = .FALSE.
293
294         DO WHILE (.NOT.DONE)
295             IORDER = IORDER + 1

```

```

296         IF(IORDER.GT.NCHANNELS) STOP 114
297         ICHLORD = ICHLORDER(IORDER)
298         IF(ICHL.EQ.ICHLORD) DONE = .TRUE.
299     END DO
300
301     IF(INITP.EQ.1) THEN
302         INTERVALS = IORDER - 1
303     ELSE
304         INTERVALS = IORDER - IORDEROLD
305         IF(INTERVALS.LE.0) INTERVALS = NCHANNELS + INTERVALS
306     END IF
307
308     TCUR = TCUR + INTERVALS * DT
309     IORDEROLD = IORDER
310
311     ILEO = ICHLLEO(ICHL)
312
313     IF(ILEO.EQ.0) THEN
314
315         J = 0
316         DONE = .FALSE.
317
318         DO WHILE (.NOT.DONE)
319             J = J + 1
320             ICHLPR = ICHLPRSEQ(J)
321             IF(ICHLPR.EQ.ICHL) THEN
322                 IF(J.LE.NLEOS) ILEO = J
323                 DONE = .TRUE.
324             END IF
325             IF(J.EQ.NCHANNELS) DONE = .TRUE.
326         END DO
327
328     END IF
329
330 C-----
331 C   In simulation mode, determine channel index and update current time. Also
332 C   determine LEO satellite index. If channel is not currently assigned a LEO
333 C   and is the highest priority free channel, then select an unassigned LEO in
334 C   sequence, if available. Note: all channel assignments are performed in
335 C   SUBROUTINE ASGNMFN.
336 C-----
337
338     ELSE
339
340         IORDER = MOD(IORDER,NCHANNELS) + 1
341         ICHL = ICHLORDER(IORDER)
342
343         IF(INITP.EQ.0) TCUR = TCUR + DT
344         TCURM = TCUR/601
345         ITCURM = NINT(TCURM)
346         IF(DABS(TCURM-ITCURM).LT.1D-6) WRITE(*,80) ITCURM
347 80     FORMAT(I7,' MIN')
348
349         IF(TCUR.GT.TSTOP) THEN
350             WRITE(*,90)
351 90     FORMAT(' END OF SIMULATION')
352             WRITE(20,95) TSTOP
353 95     FORMAT('/ TIME =',F12.3,' SEC',8X,'END OF SIMULATION')
354             STOP
355         END IF
356
357         ILEO = ICHLLEO(ICHL)
358
359         IF(ILEO.EQ.0) THEN
360
361             J = 0
362             DONE = .FALSE.
363
364             DO WHILE (.NOT.DONE)
365                 J = J + 1
366                 ICHLPR = ICHLPRSEQ(J)
367                 IF(ICHLLEO(ICHLPR).EQ.0) THEN
368                     IF(ICHLPR.EQ.ICHL) THEN
369                         I=0

```

```
370             DO WHILE (.NOT.DONE)
371                 ILEOSEQ = MOD(ILEOSEQ,NLEOS) + 1
372                 IF(LEOCHL(ILEOSEQ).EQ.0) THEN
373                     ILEO = ILEOSEQ
374                     DONE = .TRUE.
375                 END IF
376                 I = I + 1
377                 IF(I.EQ.NLEOS) DONE = .TRUE.
378             END DO
379         END IF
380         DONE = .TRUE.
381     END IF
382     IF(J.EQ.NCHANNELS) DONE = .TRUE.
383 END DO
384
385     END IF
386
387     END IF
388
389 C-----
390 C   Proceed to Navigation Function.
391 C-----
392
393     CALL NAVIGFN(HEO,LEO,NLEOS,ILEO,NCHANNELS,ICHL,LEOCHL,
394     1           ICHLLEO,TCUR)
395
396 C-----
397 C   In real-time mode, get disc step commands.
398 C-----
399
400     IF(REALTIME) THEN
401         ISTEPA = ISTEPASAVE
402         ISTEPB = ISTEPBSAVE
403     END IF
404
405     INITP = 0
406
407     RETURN
408     END
```



```
1      SUBROUTINE INV3X3(MAT)
2
3      REAL*8 MAT(3,3),TEMP(3,3),DET
4
5      DET = MAT(1,1)*MAT(2,2)*MAT(3,3) + MAT(1,2)*MAT(2,3)*MAT(3,1) +
6      &      MAT(1,3)*MAT(2,1)*MAT(3,2) - MAT(1,3)*MAT(2,2)*MAT(3,1) -
7      &      MAT(1,1)*MAT(2,3)*MAT(3,2) - MAT(1,2)*MAT(2,1)*MAT(3,3)
8
9      TEMP(1,1)=MAT(2,2)*MAT(3,3)-MAT(2,3)*MAT(3,2)
10     TEMP(1,2)=MAT(2,3)*MAT(3,1)-MAT(2,1)*MAT(3,3)
11     TEMP(1,3)=MAT(2,1)*MAT(3,2)-MAT(2,2)*MAT(3,1)
12     TEMP(2,1)=MAT(1,3)*MAT(3,2)-MAT(1,2)*MAT(3,3)
13     TEMP(2,2)=MAT(1,1)*MAT(3,3)-MAT(1,3)*MAT(3,1)
14     TEMP(2,3)=MAT(1,2)*MAT(3,1)-MAT(1,1)*MAT(3,2)
15     TEMP(3,1)=MAT(1,2)*MAT(2,3)-MAT(1,3)*MAT(2,2)
16     TEMP(3,2)=MAT(1,3)*MAT(2,1)-MAT(1,1)*MAT(2,3)
17     TEMP(3,3)=MAT(1,1)*MAT(2,2)-MAT(1,2)*MAT(2,1)
18
19     DO I=1,3
20         DO J=1,3
21             MAT(I,J)=TEMP(J,I)/DET
22         END DO
23     END DO
24
25     RETURN
26     END
```

```

1  C-----
2
3      SUBROUTINE NAVIGFN (HEO,LEO,NLEOS,ILO,NCHANNELS,ICHL,
4      1      LEOCHL,ICHLLEO,TCUR)
5
6  C   This routine simulates the Navigation Function, which consists of the
7  C   Truth Generator, Scene Processor, and Navigation Processor.
8  C
9  C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
10 C
11 C   Inputs:
12 C
13 C      HEO      = Geocentric-equatorial coordinates of HEO satellite
14 C      LEO      = Geocentric-equatorial coordinates of LEO satellites
15 C      NLEOS    = Number of LEO satellites
16 C      ILO      = LEO satellite index
17 C      NCHANNELS = Number of disc channels
18 C      ICHL     = Channel index
19 C      LEOCHL   = LEO channel assignments
20 C      ICHLLEO  = Channel LEO assignments
21 C      TCUR     = Current time
22 C      IRATE    = Tracking rate (MISCBLK)
23 C      REALTIME = Real-time flag (MISCBLK)
24 C      WFLAG    = Write flag (MISCBLK)
25 C      PRDH     = Period of HEO satellite (ORBITBLK)
26 C      PRDL     = Periods of LEO satellites (ORBITBLK)
27 C
28 C-----
29
30      IMPLICIT REAL*8 (A-H,O-Z)
31      REAL*8 HEO(6),LEO(6,6),HEONAV(6),LEONAV(6,6)
32      REAL*8 AZS(6),ELS(6)
33      INTEGER*2 LEOCHL(6),ICHLLEO(6),INITLEO(6)
34      LOGICAL ECLIPSE(6),OLDECLIPSE(6)
35      CHARACTER*1 DASH(73),ECLIND
36      INCLUDE 'CIRCBK.FCB'
37      INCLUDE 'MISCBLK.FCB'
38      INCLUDE 'ORBITBLK.FCB'
39
40      DATA ECLIPSE /6*.FALSE./
41      DATA DASH /73*'-'/
42      DATA INITLEO /6*1/
43      DATA INIT /1/
44
45 C-----
46 C   Truth Generator simulates orbital dynamics of HEO and LEO satellites.
47 C-----
48
49      IF(ILO.NE.0) THEN
50
51          IF(.NOT.REALTIME) THEN
52
53              CALL TRUTHGN(HEO,LEO,NLEOS,ILO,TCUR)
54
55              IF(WFLAG) THEN
56                  IF(INIT.EQ.1) WRITE(11,20) DASH
57                  WRITE(11,25) TCUR,(HEO(J),J=1,6)
58                  WRITE(11,26) ILO,(LEO(J,ILO),J=1,6)
59              END IF
60
61          END IF
62
63      20  FORMAT(// ' TRUTH GENERATOR',4X,'(KM)',20X,'X',12X,'Y',12X,'Z'/
64      1    20X,'(KM/SEC)',16X,2HX',11X,2HY',11X,2HZ'// ' ',73A1)
65      25  FORMAT(// ' TIME =',F12.3,' SEC',6X,'HEO ',3F13.3/35X,3F13.3)
66      26  FORMAT(/29X,'LEO',12,1X,3F13.3/35X,3F13.3)
67
68 C-----
69 C   Scene Processor determines FOV coordinates of LEO satellites.
70 C-----
71
72      IF(.NOT.REALTIME) THEN
73

```

```

74      CALL SCENEPR(HEO,LEO,NLEOS,ILEO,NCHANNELS,TCUR,
75      1          ECLIPSE,AZS,ELS)
76
77      IF(INITLEO(ILEO).EQ.1) THEN
78          IF(ECLIPSE(ILEO)) WRITE(20,50) TCUR,ILEO
79          IF(.NOT.ECLIPSE(ILEO)) WRITE(20,55) TCUR,ILEO
80      ELSE
81          IF(ECLIPSE(ILEO).AND..NOT.OLDECLIPSE(ILEO))
82      1          WRITE(20,50) TCUR,ILEO
83          IF(.NOT.ECLIPSE(ILEO).AND.OLDECLIPSE(ILEO))
84      1          WRITE(20,55) TCUR,ILEO
85      END IF
86
87      OLDECLIPSE(ILEO) = ECLIPSE(ILEO)
88
89      IF(WFLAG) THEN
90          IF(INIT.EQ.1) WRITE(12,60) (DASH(J),J=1,69)
91          ECLIND = ' '
92          IF(ECLIPSE(ILEO)) ECLIND = 'E'
93          WRITE(12,65) TCUR,ILEO,ECLIND,AZS(ILEO)*RAD,ELS(ILEO)*RAD
94      END IF
95
96      END IF
97
98      50  FORMAT(/' TIME =',F12.3,' SEC',8X,'LEO',12,' ECLIPSED')
99      55  FORMAT(/' TIME =',F12.3,' SEC',8X,'LEO',12,' NOT ECLIPSED')
100     60  FORMAT(/' SCENE PROCESSOR',22X,'ECLIPSE',5X,'AZ(DEG)',
101     1    6X,'EL(DEG)')/' ',69A1)
102     65  FORMAT(/' TIME =',F12.3,' SEC',5X,'LEO',12,A9,F15.5,F13.5)
103
104     C-----
105     C  Navigation Processor superimposes ephemeris errors.
106     C-----
107
108     IF(.NOT.REALTIME) THEN
109
110         CALL NAVIGPR(HEO,LEO,NLEOS,ILEO,TCUR,HEONAV,LEONAV)
111
112         IF(WFLAG) THEN
113             IF(INIT.EQ.1) WRITE(13,80) DASH
114             WRITE(13,25) TCUR,(HEONAV(J),J=1,6)
115             WRITE(13,26) ILEO,(LEONAV(J,ILEO),J=1,6)
116         END IF
117
118     END IF
119
120     80  FORMAT(/' NAVIGATION PROCESSOR  (KM)',16X,'X',12X,'Y',12X,'Z'
121     1    /24X,'(KM/SEC)',12X,2HX',11X,2HY',11X,2HZ')/' ',73A1)
122
123
124     INITLEO(ILEO) = 0
125     INIT = 0
126
127     END IF
128
129     C-----
130     C  Proceed to Assignment Function.
131     C-----
132
133     CALL ASGNMFN(HEONAV,LEONAV,ECLIPSE,AZS,ELS,ILEO,ICHL,LEOCHL,
134     1          ICHLLEO,TCUR)
135
136     RETURN
137     END

```

```

1  C-----
2
3      SUBROUTINE NAVIGPR (HEO,LEO,NLEOS,ILEO,TCUR,HEONAV,LEONAV)
4
5  C  This routine simulates the Navigation Processor, which superimposes
6  C  ephemeris errors on the true satellite positions and velocities.
7  C
8  C  Prototype Version 1.0      Avtec Systems, Inc.      August 1992
9  C
10 C  Entry points:
11 C
12 C      NAVIGPR      Initialization is performed in first call
13 C      NAVIGPR2     Initialization is not performed
14 C
15 C  Inputs:
16 C
17 C      HEO  = Geocentric-equatorial coordinates of HEO satellite
18 C      LEO  = Geocentric-equatorial coordinates of LEO satellites
19 C      NLEOS = Number of LEO satellites
20 C      ILEO = LEO satellite index
21 C      TCUR = Current time
22 C      PRDH = Period of HEO orbit (ORBITBLK)
23 C      PRDL = Periods of LEO orbits (ORBITBLK)
24 C
25 C  Outputs:
26 C
27 C      HEONAV      = Coordinates of HEO satellite including ephemeris errors
28 C      LEONAV(ILEO) = Coordinates of LEO satellite including ephemeris errors
29 C
30 C-----
31
32      IMPLICIT REAL*8 (A-H,O-Z)
33      REAL*8 HEO(6),LEO(6,6),HEONAV(6),LEONAV(6,6)
34      REAL*8 HPHS(3),LPHS(3,6),LFREQ(6)
35      INCLUDE 'CIRCBK.FCB'
36      INCLUDE 'ORBITBLK.FCB'
37
38      DATA INIT /1/
39
40      IF(INIT.EQ.1) THEN
41
42          DO J=1,3
43              CALL RNDM(UNIF)
44              HPHS(J) = TWOPI*UNIF
45              DO I=1,NLEOS
46                  CALL RNDM(UNIF)
47                  LPHS(J,I) = TWOPI*UNIF
48              END DO
49          END DO
50
51          HFREQ = TWOPI/PRDH
52
53          DO I=1,NLEOS
54              LFREQ(I) = TWOPI/PRDL(I)
55          END DO
56
57          INIT = 0
58
59      END IF
60
61  C-----
62      ENTRY NAVIGPR2 (HEO,LEO,ILEO,TCUR,HEONAV,LEONAV)
63  C-----
64
65      TEMPH = HFREQ*TCUR
66      TEMPL = LFREQ(ILEO)*TCUR
67
68      DO J=1,3
69
70          ARG = TEMPH+HPHS(J)
71          HEONAV(J) = HEO(J) + EPHH*DSIN(ARG)
72          HEONAV(J+3) = HEO(J+3) + EPHH*HFREQ*DCOS(ARG)
73

```

```
74      ARG = TEMPL+LPHS(J,ILEO)
75      LEONAV(J,ILEO) = LEO(J,ILEO) + EPHL*DSIN(ARG)
76      LEONAV(J+3,ILEO) = LEO(J+3,ILEO) + EPHL*LFREQ(ILEO)*DCOS(ARG)
77
78      END DO
79
80      RETURN
81      END
```

```
1 C-----
2
3     PROGRAM OMA
4
5 C   Control software simulation for the Optical Multiple Access (OMA) system.
6 C   (compatible with Microsoft FORTRAN Version 5.0 and VAX FORTRAN V5.5)
7 C
8 C   This main program simulates the real-time executive program.
9 C
10 C
11 C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
12 C
13 C
14 C   Inputs and outputs:  see SUBROUTINE INTRFACE
15 C
16 C-----
17
18     REAL*8 U,V
19     INTEGER*2 ICHANNEL
20     INTEGER*4 ISTEPA,ISTEPB
21     LOGICAL REALTIME,INTRACK
22
23     DATA REALTIME /.FALSE./
24
25 C-----
26 C
27 C   In simulation mode:
28 C
29 C     REALTIME is false.
30 C     ICHANNEL,INTRACK,U,V,ISTEPA,ISTEPB are not used.
31 C
32 C-----
33
34     CALL INIT-REALTIME)
35
36     10 CALL POSITION(ICHANNEL,INTRACK,U,V,ISTEPA,ISTEPB)
37     GO TO 10
38
39     END
```

```

1  C-----
2
3      SUBROUTINE QFIT(HINTG,LINTG,NLEOS,T1,DTINTG)
4
5  C   This routine fits a quadratic function of time to each position coordinate
6  C   and velocity component using 3 points.
7  C
8  C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
9  C
10 C   Inputs:
11 C
12 C       HINTG = 3 Runge-Kutta integration points of HEO satellite state
13 C       LINTG = 3 Runge-Kutta integration points of LEO satellite states
14 C       NLEOS = Number of LEO satellites
15 C       T1    = Time of first integration point
16 C       DTINTG = Integration time interval
17 C
18 C-----
19
20      IMPLICIT REAL*8 (A-H,O-Z)
21      REAL*8 HEO(6),LEO(6,6),HINTG(6,3),LINTG(6,6,3)
22      REAL*8 TMAT(3,3),HCOEFF(6,3),LCOEFF(6,6,3)
23
24      TMAT(1,2) = T1
25      TMAT(2,2) = T1 + DTINTG
26      TMAT(3,2) = T1 + 2*DTINTG
27
28      DO J=1,3
29          TMAT(J,1) = 1D0
30          TMAT(J,3) = TMAT(J,2)**2
31      END DO
32
33      CALL INV3X3(TMAT)
34
35      DO ICOORD=1,6
36          DO ITERM=1,3
37
38              TEMP = 0
39              DO J=1,3
40                  TEMP = TEMP + HINTG(ICOORD,J)*TMAT(ITERM,J)
41              END DO
42              HCOEFF(ICOORD,ITERM) = TEMP
43
44              DO I=1,NLEOS
45                  TEMP = 0
46                  DO J=1,3
47                      TEMP = TEMP + LINTG(ICOORD,I,J)*TMAT(ITERM,J)
48                  END DO
49                  LCOEFF(ICOORD,I,ITERM) = TEMP
50              END DO
51
52          END DO
53      END DO
54
55      RETURN
56
57 C-----
58
59      ENTRY QVAL(TCUR,ILEO,HEO,LEO)
60
61  C   Evaluate quadratic functions of position coordinates and velocity
62  C   components.
63  C
64  C
65  C   Inputs:
66  C
67  C       TCUR = Current time
68  C       ILEO = LEO satellite index
69  C
70  C   Outputs:
71  C
72  C       HEO = Geocentric-equatorial coordinates of HEO satellite
73  C       LEO = Geocentric-equatorial coordinates of LEO satellites

```

```
74 C
75 C-----
76
77      TCURSQ = TCUR**2
78
79      DO ICOORD=1,6
80
81          HEO(ICOORD) = HCOEFF(ICOORD,1) + HCOEFF(ICOORD,2)*TCUR +
82      1      HCOEFF(ICOORD,3)*TCURSQ
83
84          LEO(ICOORD,ILEO) = LCOEFF(ICOORD,ILEO,1) +
85      1      LCOEFF(ICOORD,ILEO,2)*TCUR + LCOEFF(ICOORD,ILEO,3)*TCURSQ
86
87      END DO
88
89      RETURN
90      END
```



```
1 C-----
2
3 SUBROUTINE RNDM (Z)
4
5 C GENERATES RANDOM NUMBERS HAVING A UNIFORM DISTRIBUTION, USING THE
6 C MULTIPLICATIVE CONGRUENTIAL METHOD
7 C
8 C REFERENCE:
9 C B. CARNAHAN, H.A. LUTHER AND J.D. WILKES, "APPLIED NUMERICAL METHODS"
10 C JOHN WILEY & SONS, NEW YORK, NY, 1969, pp.545-549.
11 C
12 C-----
13
14 IMPLICIT REAL*8 (A-H,O-Z)
15 INTEGER*4 A,M,X
16
17 DATA I /1/
18
19 IF(I .EQ. 1) THEN
20 I = 0
21 M = 1048576 !2**20
22 FM = M
23 X = 566387 !SEED
24 A = 1027 !2**10+3
25 END IF
26
27 X = MOD(A*X,M)
28 Z = X/FM
29 RETURN
30 END
```

```
1 C-----
2
3     SUBROUTINE RUK(X,DT,XNEW)
4
5 C   Fourth order Runge-Kutta integrator.
6 C
7 C   Courtesy of Bob Dasenbrock of NRL. Modified for OMA.
8 C
9 C   Inputs:
10 C
11 C       X(1,2,3) = current satellite position in geocentric coordinates (km)
12 C       X(4,5,6) = current satellite velocity in geocentric coordinates (km/sec)
13 C       DT       = propagation time interval (sec)
14 C
15 C   Outputs:
16 C
17 C       XNEW(1,2,3) = satellite position propagated ahead DT sec
18 C       XNEW(4,5,6) = satellite velocity propagated ahead DT sec
19 C
20 C-----
21
22     IMPLICIT REAL*8 (A-H,O-Z)
23     DIMENSION D(6),F(6),U(6),X(6),XNEW(6)
24
25     CALL DFQ(X,D,DT)
26
27     DO K=1,6
28         U(K)=X(K)+0.5D0*D(K)
29     END DO
30
31     CALL DFQ(U,F,DT)
32
33     DO K=1,6
34         D(K)=D(K)+2.0D0*F(K)
35         U(K)=X(K)+0.5D0*F(K)
36     END DO
37
38     CALL DFQ(U,F,DT)
39
40     DO K=1,6
41         D(K)=D(K)+2.0D0*F(K)
42         U(K)=X(K)+F(K)
43     END DO
44
45     CALL DFQ(U,F,DT)
46
47     DO K=1,6
48         XNEW(K)=X(K)+(D(K)+F(K))/6.0D0
49     END DO
50
51     RETURN
52     END
```

```

1  C-----
2
3      SUBROUTINE SCANGN (ILEO,ICHL,MODE,SCANRES,TCUR,AZSCAN,ELSCAN)
4
5  C    Scan Generator implements acquisition and reacquisition patterns.
6  C
7  C    Prototype Version 1.0      Avtec Systems, Inc.      August 1992
8  C
9  C    Inputs:
10 C
11 C        ILEO      = LEO satellite index
12 C        ICHL      = Channel index
13 C        MODE      = Channel modes
14 C        SCANRES   = Scan reset flags
15 C        BEAMW     = HEO beamwidth
16 C        TCUR      = Current time
17 C        TSMALLSCAN = Earliest time for small scan pattern (ATTBLK)
18 C        WFLAG     = Write flag (MISCBLK)
19 C
20 C    Outputs:
21 C
22 C        AZSCAN(ILEO) = Azimuth coordinate of scan position
23 C        ELSCAN(ILEO) = Elevation coordinate of scan position
24 C        MODE(ICHL)   = Channel mode
25 C
26 C-----
27
28      IMPLICIT REAL*8 (A-H,O-Z)
29      REAL*8 AZSCAN(6),ELSCAN(6)
30      INTEGER*2 IPOSIT(6),ICOUNT(6),ISIZE(6),ISIZE2(6),ISIGN(6)
31      LOGICAL SCANRES(6),SMALLSCAN
32      CHARACTER*13 MODE(6)
33      INCLUDE 'ATTBLK.FCB'
34      INCLUDE 'FOVBLK.FCB'
35      INCLUDE 'MISCBLK.FCB'
36
37      DATA ISQUARE /1089/           ! Number of beams in square pattern
38      DATA ISMALLSQUARE /121/       ! Number of beams in small square pattern
39      DATA SMALLSCAN /.FALSE./
40      DATA IPOSIT /6*0/
41      DATA INIT /1/
42
43      IF(INIT.EQ.1) THEN
44          HALFBEAM = BEAMW/2
45          QTRBEAM = BEAMW/4
46          INIT = 0
47      END IF
48
49      IF(.NOT.SMALLSCAN .AND. TCUR.GE.TSMALLSCAN) THEN
50          ISQUARE = ISMALLSQUARE
51          SMALLSCAN = .TRUE.
52      END IF
53
54  C-----
55  C    If the scan pattern was previously just finished, repeat it.
56  C-----
57
58      IF(IPOSIT(ILEO).GE.ISQUARE .AND. .NOT.SCANRES(ILEO)) THEN
59
60          SCANRES(ILEO) = .TRUE.
61          IF(WFLAG) WRITE(15,20)
62          20  FORMAT(17X,'REPEAT SCAN')
63          WRITE(20,21) TCUR,ICHL
64          21  FORMAT(/' TIME =' ,F12.3, ' SEC',8X,'CHANNEL',12, ' REPEAT SCAN')
65
66          IF(MODE(ICHL).EQ.'ACQUISITION') THEN
67              MODE(ICHL) = 'SLEW TO ACQ'
68          ELSE
69              MODE(ICHL) = 'SLEW TO REACQ'
70          END IF
71
72          IF(WFLAG) WRITE(15,23) MODE(ICHL)
73          23  FORMAT(A16)

```

```
74
75     END IF
76
77 C-----
78 C   If scan reset flag is true, initialize the scan pattern.
79 C-----
80
81     IF(SCANRES(ILEO)) THEN
82
83         IPOSIT(ILEO) = 1
84         ICOUNT(ILEO) = 0
85         ISIZE(ILEO) = 1
86         ISIZE2(ILEO) = 2
87         ISIGN(ILEO) = 1
88         SCANRES(ILEO) = .FALSE.
89
90         IF(ISQUARE .EQ. ISQUARE/2 * 2) THEN
91             AZSCAN(ILEO) = -QRTBEAM
92             ELSCAN(ILEO) = -QRTBEAM
93         ELSE
94             AZSCAN(ILEO) = 0
95             ELSCAN(ILEO) = 0
96         END IF
97     ELSE
98
99
100 C-----
101 C   If scan reset flag is false, continue the scan pattern.
102 C-----
103
104         IPOSIT(ILEO) = IPOSIT(ILEO) + 1
105         ICOUNT(ILEO) = ICOUNT(ILEO) + 1
106
107         IF(ICOUNT(ILEO) .LE. ISIZE(ILEO)) THEN
108             ELSCAN(ILEO) = ELSCAN(ILEO) + HALFBEAM*ISIGN(ILEO)
109         ELSE
110             AZSCAN(ILEO) = AZSCAN(ILEO) + HALFBEAM*ISIGN(ILEO)
111         END IF
112
113         IF(ICOUNT(ILEO) .EQ. ISIZE2(ILEO)) THEN
114             ICOUNT(ILEO) = 0
115             ISIZE(ILEO) = ISIZE(ILEO) + 1
116             ISIZE2(ILEO) = ISIZE(ILEO) * 2
117             ISIGN(ILEO) = - ISIGN(ILEO)
118         END IF
119
120     END IF
121
122     IF(WFLAG) WRITE(15,50) IPOSIT(ILEO)
123 50 FORMAT(17X,'SCANPOS =',I10)
124
125     RETURN
126     END
```

```

1  C-----
2
3      SUBROUTINE SCENEPR (HEO,LEO,NLEOS,ILEO,NCHANNELS,TCUR,
4      1                      ECLIPSE,AZS,ELS)
5
6  C   The Scene Processor computes the azimuth & elevation of a received
7  C   LEO beam center, including the effects of attitude errors.
8  C
9  C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
10 C
11 C   Inputs:
12 C
13 C       HEO      = Geocentric-equatorial coordinates of HEO satellite
14 C       LEO      = Geocentric-equatorial coordinates of LEO satellites
15 C       NLEOS    = Number of LEO satellites
16 C       ILEO     = LEO satellite index
17 C       NCHANNELS = Number of disc channels
18 C       TCUR     = Current time
19 C       IRATE    = Tracking rate (MISCBLK)
20 C       PRDH     = Period of HEO orbit (ORBITBLK)
21 C
22 C   Outputs:
23 C
24 C       ECLIPSE(ILEO) = True if LEO is eclipsed by the earth
25 C       AZS(ILEO)    = Azimuth in HEO FOV
26 C       ELS(ILEO)    = Elevation in HEO FOV
27 C
28 C-----
29
30      IMPLICIT REAL*8 (A-H,O-Z)
31      REAL*8 HEO(6),LEO(6,6),AZS(6),ELS(6),AZAHEAD(6),ELAHEAD(6)
32      LOGICAL ECLIPSE(6),FINDADJT,DONE
33      INCLUDE 'ATTBLK.FCB'
34      INCLUDE 'CIRCBK.FCB'
35      INCLUDE 'MISCBLK.FCB'
36      INCLUDE 'ORBITBLK.FCB'
37
38      DATA ATTRP      /1D-3/      ! Slow roll and pitch variation (rad)
39      DATA ATTRPADJ /2.5D-4/      ! Adjusted slow roll and pitch variation (rad)
40      DATA ATTY       /3D-3/      ! Slow yaw variation (rad)
41      DATA ATTYADJ    /7.5D-4/      ! Adjusted slow yaw variation (rad)
42      DATA VIBSIGMA /1.585D-5/      ! Standard deviation of vibration (rad)
43      DATA VIBBANDW /1D0/          ! 3 dB bandwidth of vibration PSD (Hz)
44
45      DATA FINDADJT /.FALSE./
46      DATA ROLADJT /1D30/, PCHADJT /1D30/, YAWADJT /1D30/
47      DATA OLDVIBROL /0D0/, OLDVIBPCH /0D0/, OLDVIBYAW /0D0/
48      DATA INIT /1/
49
50      IF(INIT.EQ.1) THEN
51          C1 = TWOPI / (PRDH/2)
52          C2 = 1 - DEXP(-TWOPI*VIBBANDW/(IRATE*NCHANNELS)) ! All channels called
53          GVAR = (2-C2)/C2 * VIBSIGMA**2
54          CALL RNDM(UNIF)
55          ROLPHS = TWOPI*UNIF
56          CALL RNDM(UNIF)
57          PCHPHS = TWOPI*UNIF
58          CALL RNDM(UNIF)
59          YAWPHS = TWOPI*UNIF
60          IF(NLEOS.GE.4 .AND. NCHANNELS.GE.4) FINDADJT = .TRUE.
61          INIT = 0
62      END IF
63
64  C-----
65  C   Generate random vibrations in roll, pitch and yaw.
66  C-----
67
68      VIBROL = OLDVIBROL + (GAUSS(0D0,GVAR)-OLDVIBROL) * C2
69      VIBPCH = OLDVIBPCH + (GAUSS(0D0,GVAR)-OLDVIBPCH) * C2
70      VIBYAW = OLDVIBYAW + (GAUSS(0D0,GVAR)-OLDVIBYAW) * C2
71
72  C-----
73  C   If #acquisitions is at least 2 (and there are at least 4 LEOs and channels),

```

```
74 C determine when to adjust variations in roll, pitch and yaw.
75 C-----
76
77     TEMP = C1*TCUR
78
79     IF(FINDADJT .AND. NACQS.GE.2) THEN
80
81         ROLADJT = - ROLPHS
82         DONE = .FALSE.
83         DO WHILE (.NOT.DONE)
84             ROLADJT = ROLADJT + PI
85             IF(ROLADJT.GE.TEMP) DONE = .TRUE.
86         END DO
87
88         PCHADJT = - PCHPHS
89         DONE = .FALSE.
90         DO WHILE (.NOT.DONE)
91             PCHADJT = PCHADJT + PI
92             IF(PCHADJT.GE.TEMP) DONE = .TRUE.
93         END DO
94
95         YAWADJT = - YAWPHS
96         DONE = .FALSE.
97         DO WHILE (.NOT.DONE)
98             YAWADJT = YAWADJT + PI
99             IF(YAWADJT.GE.TEMP) DONE = .TRUE.
100        END DO
101
102        TSMALLSCAN = DMAX1(ROLADJT,PCHADJT,YAWADJT) / C1
103        FINDADJT = .FALSE.
104
105    END IF
106
107 C-----
108 C Determine total errors in roll, pitch and yaw. If time is large enough,
109 C adjust attitude variations.
110 C-----
111
112     IF(TEMP.LT.ROLADJT) THEN
113         ROLERR = ATTRP * DSIN(TEMP+ROLPHS) + VIBROL
114     ELSE
115         ROLERR = ATTRPADJ * DSIN(TEMP+ROLPHS) + VIBROL
116     END IF
117
118     IF(TEMP.LT.PCHADJT) THEN
119         PCHERR = ATTRP * DSIN(TEMP+PCHPHS) + VIBPCH
120     ELSE
121         PCHERR = ATTRPADJ * DSIN(TEMP+PCHPHS) + VIBPCH
122     END IF
123
124     IF(TEMP.LT.YAWADJT) THEN
125         YAWERR = ATTY * DSIN(TEMP+YAWPHS) + VIBYAW
126     ELSE
127         YAWERR = ATTYADJ * DSIN(TEMP+YAWPHS) + VIBYAW
128     END IF
129
130 C-----
131 C Calculate azimuth and elevation of received LEO beam center.
132 C-----
133
134     CALL XYZAZEL(HEO,LEO,ILEO,'T','R',ECLIPSE,AZS,ELS,AZAHEAD,ELAHEAD)
135
136 C-----
137 C Include effects of attitude errors on azimuth and elevation.
138 C-----
139
140     AZTEMP = AZS(ILEO) + PCHERR
141     ELTEMP = ELS(ILEO) + ROLERR
142
143     SINY = DSIN(YAWERR)
144     COSY = DCOS(YAWERR)
145
146     AZS(ILEO) = AZTEMP*COSY-ELTEMP*SINY
147     ELS(ILEO) = AZTEMP*SINY+ELTEMP*COSY
```

148

149 OLDVIBROL = VIBROL

150 OLDVIBPCH = VIBPCH

151 OLDVIBYAW = VIBYAW

152

153 RETURN

154 END

```
1 C-----
2
3     SUBROUTINE TRUTHGN (HEO,LEO,NLEOS,ILEO,TCUR)
4
5 C   Truth Generator simulates orbital dynamics of the HEO and LEO satellites.
6 C
7 C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
8 C
9 C   Inputs:
10 C
11 C       HEO  = Geocentric-equatorial coordinates of HEO satellite
12 C       LEO  = Geocentric-equatorial coordinates of LEO satellites
13 C       NLEOS = Number of LEO satellites
14 C       ILEO = LEO satellite index
15 C       TCUR  = Current time
16 C
17 C   Outputs:
18 C
19 C       HEO  = Updated geocentric-equatorial coordinates of HEO satellite
20 C       LEO  = Updated geocentric-equatorial coordinates of LEO satellites
21 C
22 C-----
23
24     IMPLICIT REAL*8 (A-H,O-Z)
25     REAL*8 HEO(6),LEO(6,6),HINTG(6,3),LINTG(6,6,3)
26     DATA DTINTG /1D1/
27     DATA INIT /1/
28
29 C-----
30 C   If initializing, propagate orbits using Runge-Kutta integration to obtain
31 C   three points for quadratic fit.
32 C-----
33
34     IF (INIT.EQ.1) THEN
35
36         CALL COPYVEC(HEO,6,HINTG(1,1))
37         CALL RUK(HEO,DTINTG,HINTG(1,2))
38         CALL RUK(HINTG(1,2),DTINTG,HINTG(1,3))
39
40         DO I=1,NLEOS
41             CALL COPYVEC(LEO(1,I),6,LINTG(1,I,1))
42             CALL RUK(LEO(1,I),DTINTG,LINTG(1,I,2))
43             CALL RUK(LINTG(1,I,2),DTINTG,LINTG(1,I,3))
44         END DO
45
46         CALL QFIT(HINTG,LINTG,NLEOS,ODO,DTINTG)
47         TPROP = DTINTG
48         INIT = 0
49
50 C-----
51 C   If necessary to obtain a later point for quadratic fit, propagate orbits
52 C   further using Runge-Kutta integration.
53 C-----
54
55     ELSE IF (TCUR.GE.TPROP) THEN
56
57         CALL COPYVEC(HINTG(1,2),6,HINTG(1,1))
58         CALL COPYVEC(HINTG(1,3),6,HINTG(1,2))
59         CALL RUK(HINTG(1,2),DTINTG,HINTG(1,3))
60
61         DO I=1,NLEOS
62             CALL COPYVEC(LINTG(1,I,2),6,LINTG(1,I,1))
63             CALL COPYVEC(LINTG(1,I,3),6,LINTG(1,I,2))
64             CALL RUK(LINTG(1,I,2),DTINTG,LINTG(1,I,3))
65         END DO
66
67         CALL QFIT(HINTG,LINTG,NLEOS,TPROP,DTINTG)
68         TPROP = TPROP + DTINTG
69
70     END IF
71
72 C-----
73 C   Evaluate quadratic functions to obtain satellite coordinates between
```


74 C integration points.

75 C-----

76

77 CALL QVAL(TCUR,ILEO,HEO,LEO)

78

79 RETURN

80 END

```
1      SUBROUTINE UNIT(V)
2      REAL*8 V(3),VLEN
3      VLEN=DSQRT(V(1)**2+V(2)**2+V(3)**2)
4      IF(VLEN.EQ.0) STOP 113
5      V(1)=V(1)/VLEN
6      V(2)=V(2)/VLEN
7      V(3)=V(3)/VLEN
8      RETURN
9      END
```

```
1      REAL*8 FUNCTION VLEN(V)
2      REAL*8 V(3)
3      VLEN=DSQRT(V(1)**2+V(2)**2+V(3)**2)
4      RETURN
5      END
```

```
1      DOUBLE PRECISION FUNCTION XKEP (ECC,XM,TOL)
2      C      SOLVES KEPLER'S EQUATION "XM=XKEP-ECC*SIN(XKEP)"
3      C      SOLVES FOR "XKEP" IN RADIANs GIVEN "XM" AND "ECC"
4      IMPLICIT REAL*8 (A-H,O-Z)
5      EOLD=XM
6      DO 10 K=1,100
7          SEC=DSIN(EOLD)*ECC
8          CEC=DCOS(EOLD)*ECC
9          ENEW=(XM+SEC-EOLD*CEC)/(1.000-CEC)
10         DE=DABS(ENEW-EOLD)
11         IF (DE.LE.TOL) GO TO 20
12     10 EOLD=ENEW
13     WRITE(*,100)
14 100 FORMAT (10X,' ** KEPLER'S EQUATION DID NOT CONVERGE **')
15     STOP
16 20 XKEP=ENEW
17     RETURN
18     END
```

```

1  C-----
2
3      SUBROUTINE XYZAZEL (HEO,LEO,ILEO,COORD,BEAM,ECLIPSE,AZ,EL,
4      1                      AZAHEAD,ELAHEAD)
5
6  C   This routine transforms LEO X,Y,Z to HEO FOV coordinates AZ,EL.
7  C
8  C   Prototype Version 1.0      Avtec Systems, Inc.      August 1992
9  C
10 C   Inputs:
11 C
12 C       HEO = Geocentric-equatorial coordinates of HEO satellite
13 C       LEO = Geocentric-equatorial coordinates of LEO satellites
14 C       ILEO = LEO satellite index
15 C       COORD = Coord source: 'T' (Truth Generator) or 'N' (Navigation Processor)
16 C       BEAM = Beam direction: 'R' (receive) or 'T' (transmit)
17 C
18 C   Outputs:
19 C
20 C       ECLIPSE(ILEO) = Eclipse flag
21 C       AZ(ILEO)      = Azimuth
22 C       EL(ILEO)      = Elevation
23 C       AZAHEAD(ILEO) = Azimuth point-ahead angle
24 C       ELAHEAD(ILEO) = Elevation point-ahead angle
25 C
26 C-----
27
28      IMPLICIT REAL*8 (A-H,O-Z)
29      REAL*8 HEO(6),LEO(6,6),AZ(6),EL(6),AZAHEAD(6),ELAHEAD(6)
30      REAL*8 AZVEC(3),ELVEC(3),DOWN(3),TOLEO(3)
31      LOGICAL ECLIPSE(6)
32      CHARACTER*1 COORD,BEAM
33      INCLUDE 'ORBITBLK.FCB'
34
35      DATA C /2.99792458D5/      ! Speed of light (km/sec)
36      DATA RE /6.378145D3/      ! Earth radius (km)
37      DATA INIT /1/
38
39      IF(INIT.EQ.1) THEN
40          RESQ = RE**2
41          RSQ = (RE+DSQRT(3D0)*EPHL)**2
42          INIT = 0
43      END IF
44
45      IF(BEAM .EQ. 'R') SIGN = -1
46      IF(BEAM .EQ. 'T') SIGN = 1
47
48      DO J=1,3
49          DOWN(J) = -HEO(J)
50          AZVEC(J) = HEO(J+3)      ! Circular orbit
51      END DO
52
53      CALL UNIT(DOWN)
54      CALL UNIT(AZVEC)
55      CALL CROSS(AZVEC,DOWN,ELVEC)
56
57      RHEO = VLEN(HEO)
58      IF(COORD.EQ.'T') RHORIZ = DSQRT(RHEO**2-RESQ)
59      IF(COORD.EQ.'N') RHORIZ = DSQRT(RHEO**2-RSQ)
60      COSTHRES = RHORIZ/RHEO
61
62      DO J=1,3
63          TOLEO(J) = LEO(J,ILEO) - HEO(J)
64      END DO
65
66      RTOLEO = VLEN(TOLEO)
67      CALL UNIT(TOLEO)
68
69      AZCOMP = DOT(TOLEO,AZVEC)
70      ELCOMP = DOT(TOLEO,ELVEC)
71      DNCOMP = DOT(TOLEO,DOWN)
72      IF(DNCOMP.LE.0) STOP 117
73

```

```
74      IF(DNCOMP.GE.COSTHRES .AND. RTOLEO.GE.RHORIZ) THEN
75          ECLIPSE(ILEO) = .TRUE.
76      ELSE
77          ECLIPSE(ILEO) = .FALSE.
78      END IF
79
80      AZTEMP = DATAN(AZCOMP/DNCOMP)
81      ELTEMP = DATAN(ELCOMP/DNCOMP)
82
83      TPROP = DSQRT((HEO(1)-LEO(1,ILEO))**2 + (HEO(2)-LEO(2,ILEO))**2
84      1      + (HEO(3)-LEO(3,ILEO))**2) / C * SIGN
85
86      DO J=1,3
87          TOLEO(J) = LEO(J,ILEO)+TPROP*(LEO(J+3,ILEO)-HEO(J+3))-HEO(J)
88      END DO
89
90      CALL UNIT(TOLEO)
91
92      AZCOMP = DOT(TOLEO,AZVEC)
93      ELCOMP = DOT(TOLEO,ELVEC)
94      DNCOMP = DOT(TOLEO,DOWN)
95      IF(DNCOMP.LE.0) STOP 117
96
97      AZ(ILEO) = DATAN(AZCOMP/DNCOMP)
98      EL(ILEO) = DATAN(ELCOMP/DNCOMP)
99
100     AZAHEAD(ILEO) = 2 * (AZ(ILEO) - AZTEMP) * SIGN
101     ELAHEAD(ILEO) = 2 * (EL(ILEO) - ELTEMP) * SIGN
102
103     RETURN
104     END
```